Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Институт радиоэлектроники и информационных технологий — РТФ

А. А. ТИТАЕВ

ПРОМЫШЛЕННЫЕ СЕТИ

Учебное пособие

Министерство науки и высшего образования
Российской Федерации
Уральский федеральный университет
имени Первого Президента России Б. Н. Ельцина

А. А. Титаев

ПРОМЫШЛЕННЫЕ СЕТИ

Учебное пособие

Рекомендовано методическим советом
Уральского федерального университета
для студентов вуза, обучающихся
по направлениям подготовки
27.03.04 — Управление в технических системах,
09.03.01 — Информатика и вычислительная техника,
09.03.04 — Программная инженерия

Екатеринбург
Издательство Уральского университета
2020

УДК 681.5(075.8)
ББК 32.965я73
Т45

Рецензенты:
кафедра физики Уральского государственного горного университета (зав-кафедрой д-р физ.-мат. наук, проф. *И. Г. Коршунов*);
старший научный сотрудник Института теплофизики УрО РАН, канд. физ.-мат. наук, *А. А. Старостин*

Научный редактор — канд. техн. наук., доц. *В. И. Паутов*

Титаев, А. А.
Т45 Промышленные сети : учеб. пособие / А. А. Титаев ; Мин-во науки и высшего образования РФ. — Екатеринбург: Изд-во Урал. ун-та, 2020. — 124 с.

ISBN 978-5-7996-2985-4

В пособии рассмотрены интерфейсы и протоколы промышленных сетей, применяемые в области автоматизации технологических процессов и производств. Для каждой из приведенных технологий даны описание, характеристики, области применения. Приведены примеры программирования обмена между устройствами с использованием рассмотренных протоколов на языках C++ и C#.

Библиогр.: 21 назв. Табл. 15. Рис. 33.

УДК 681.5(075.8)
ББК 32.965я73

Учебное издание

Титаев Александр Анатольевич

ПРОМЫШЛЕННЫЕ СЕТИ

Редактор О. С. Смирнова
Верстка О. П. Игнатьевой

Подписано в печать 25.02.2020. Формат 70×100/16.
Бумага офсетная. Цифровая печать. Усл. печ. л. 10,0.
Уч.-изд. л. 6,3. Тираж 30 экз. Заказ 43.

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ. 620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: +7 (343) 375-48-25, 375-46-85, 374-19-41. E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620083, Екатеринбург, ул. Тургенева, 4. Тел.: +7 (343) 358-93-06, 350-58-20, 350-90-13
Факс: +7 (343) 358-93-06. http://print.urfu.ru

ISBN 978-5-7996-2985-4

Содержание

Предисловие

Данное учебное пособие охватывает не только общетехнические аспекты сетей передачи данных, но и сконцентрировано на темах применения сетей в промышленной автоматизации технических систем. В связи с этим студентам специальности «Управление и информатика в технических системах» требуются знания специфических протоколов и интерфейсов, реализующих передачу данных в режимах, близких к режимам реального времени.

Структурно пособие состоит из шести разделов: в первом разделе дается общий обзор существующих технологий, использующихся при построении сетей промышленной автоматики; второй раздел посвящен физическим технологиям передачи данных, их преимуществам и недостаткам; разделы с третьего по пятый описывают наиболее часто используемые протоколы, позволяющие организовать обмен данными по сети с учетом требований к проектируемой системе. Раздел шесть посвящен перспективному аппаратно-независимому протоколу OPC.

Содержание пособия охватывает следующие темы: характеристики линии связи; протоколы физического уровня для передачи данных по двухпроводной линии связи (RS-232 и RS-485); протоколы физического уровня на базе группы стандартов Ethernet, включая медные и оптические носители; логические протоколы, базирующиеся на физическом стандарте RS-485 (MODBUS, PROFIBUS); логические протоколы, поддерживающие передачу информации по линиям Ethernet (MODBUS TCP, PROFINET, RT Ethernet); аппаратно-независимый протокол сбора данных OPC.

Последовательность и полнота изложения собранного в пособии материала были опробованы при чтении курса «Информационные сети и телекоммуникации».

1. Общие сведения о технологиях передачи данных

1.1. История развития сетей

Область автоматизации технологических процессов активно развивается на протяжении последних десятилетий. Исторически первыми появились локальные аналоговые системы управления, получающие данные с датчиков в виде аналоговых значений, преобразованных в электромагнитную форму (ток, напряжение), и работающие независимо друг от друга. Задача передачи данных в этом случае была тривиальной и не требовала специфических решений. По мере усложнения систем автоматики возникло сразу несколько потребностей:

- сбор информации с большого количества разнесенных в пространстве датчиков. Классический подход требовал бы прокладки линии от каждого из датчиков к управляющему устройству, что увеличивало протяженность кабельных трасс и усложняло их монтаж;
- взаимодействие нескольких управляющих устройств между собой путем обмена информацией;
- построение иерархических АСУ ТП, включающих как уровень управляющих устройств, так и верхний уровень управления с участием человека [1, с. 7].

Эти потребности привели к возникновению сетей обмена данными, ориентированных на использование в промышленной автоматике (Fieldbus). Развитие таких сетей идет параллельно с развитием локальных сетей общего назначения, и зачастую заимствует из них некоторые наиболее успешные технологии.

Одним из ранних стандартов для обмена данными является принятый Международной электротехнической комиссией (МЭК —

IEC) в 1960 г. стандарт RS-232 для соединения типа «точка-точка» двух устройств по последовательному асинхронному каналу данных. Несмотря на долгий срок жизни, данный стандарт до сих пор используется в простых случаях для соединения устройств промышленной автоматики.

Первые централизованные попытки стандартизации промышленных сетей были предприняты в 1984 г. в виде проекта стандарта IEC 61158, в котором определялись требования для открытой промышленной сети, устройств удаленного ввода/вывода, контроллеров, согласующих устройств. Однако принятие этого стандарта состоялось только в 2003 году. В 1989 г. организацией BMBF (Германское федеральное министерство по исследованиям и технологии) была разработана спецификация открытой промышленной сети, получившей название PROFIBUS (PROcess Field BUS), в основу которой легла модель ISO/OSI, принятая ранее. Позднее эту спецификацию использовал в производстве своего оборудования немецкий концерн Siemens, что позволило ей выйти на мировой рынок. В 1991 г. спецификация PROFIBUS получила статус немецкого национального стандарта DIN 19245, а позднее вошла как часть в IEC 61158. В 1996 г. стандарт PROFIBUS был оформлен в виде европейского стандарта промышленной сети (European Fieldbus Standard) EN 50170. В связи с распространением Ethernet-технологий был создан консорциум фирм, поставивший цель разработать аналог протокола PROFIBUS для Ethernet-сетей. Такой протокол получил название PROFINET и вошел в одну из редакций стандарта IEC 61158. Стандарт определяет работу протоколов на физическом, канальном и прикладном уровнях модели OSI.

Параллельно стандарту PROFIBUS шло развитие других протоколов для промышленной автоматизации. В 1979 г. фирмой Modicon (впоследствии Schneider Electric) был представлен стандарт MODBUS для обмена в режиме “ведущий-ведомый” между производимыми ею ПЛК. Впоследствии фирма открыла спецификацию протокола для всех желающих, что способствовало повышению популярности его использования вплоть до настоящего времени. Наряду с MODBUS широко используются другие стандарты (CANBUS, BitBUS, ASI).

Для взаимодействия компьютеров под управлением Windows с промышленными устройствами (обычно ПЛК) в 1996 г. был разработан стандарт OPC (OLE for Process Control). Для поддержки стандарта была создана организация OPC Foundation, включающая в себя

производителей промышленного автоматизирующего оборудования. Основное преимущество OPC состоит в том, что за счет открытости стандарта компьютер под управлением ОС Windows может получать данные с контроллеров различных производителей, поддерживающих этот стандарт. Преимущественно используется для построения систем верхнего уровня автоматизации (SCADA-систем). Первые версии стандарта были основаны на разработанных Microsoft технологиях OLE (Object Linking and Embedding), COM, DCOM. Однако с целью устранения зависимости стандарта от продуктов компании Microsoft с 2010 г. развивается спецификация OPC UA (Unified Architecture), закрепленная в стандарте IEC 62541.

В целом необходимо отметить, что, в отличие от сетей общего назначения, стандартизация сетей промышленной автоматики еще не закончила свое формирование и развивается под давлением отдельных групп компаний-производителей, отстаивающих свои интересы при разработке международных стандартов.

1.2. Модель взаимодействия открытых систем (МВОС)

В процессе развития сетевых технологий возникла потребность в едином иерархическом стандарте, описывающем взаимодействие по сети программ и устройств, выпущенных разными производителями. Ответом на эту потребность стала разработанная в 1978 г. Международной организацией по стандартизации (International Standard Organization) модель взаимодействия открытых систем (МВОС, Open System Interconnection, OSI model) [2, с. 124; 3, с. 57], описывающая общую концепцию взаимодействия сетевых устройств и технологий. Модель состоит из семи уровней, начиная от наиболее приближенного к физической среде передачи физического уровня и заканчивая наиболее удаленным от среды прикладным уровнем. В табл. 1 представлены все уровни.

На каждом из узлов, подключенных к сети, имеется программно-аппаратное обеспечение, реализующее работу на каждом из уровней модели OSI. При необходимости передачи данных они проходят каждый из уровней, начиная с седьмого и кончая первым на узле-источнике. При этом на каждом уровне полезные данные дополняются служеб-

ной информацией, специфичной для этого уровня. После прохождения физического уровня данные преобразуются оборудованием в последовательность бит и передаются в линию связи. В узле-приемнике полученные из линии связи данные передаются с физического на вышележащие уровни (в обратной последовательности). Схематический пример такой передачи представлен на рис. 1.

Таблица 1

Уровни модели МВОС (OSI)

№ уровня	Название	Описание
7	Прикладной	Решение прикладных задач (обмен файлами, почта, web и т. д.)
6	Представления	Изменение представления данных, включая кодирование, шифрование, сжатие
5	Сеансовый	Управление сеансом обмена данными
4	Транспортный	Надежная передача последовательности данных между узлами включая сегментацию, подтверждение данных.
3	Сетевой	Построение сети с многими узлами, включая адресацию и маршрутизацию.
2	Канальный	Надежная передача между двумя узлами, соединенными одной физической средой передачи
1	Физический	Прием и передача данных с учетом физической природы среды передачи

Рис. 1. Иллюстрирующий пример взаимодействия уровней модели OSI при передаче данных

Совокупность протоколов, описывающих обмен данными на всех этих уровнях, называется стеком протоколов.

Промышленные протоколы связи, использующие специфическую физическую среду, как правило, определяют взаимодействие на 1, 2 и 7 уровнях [4, с. 11]. Остальные уровни не применяются ввиду простоты задачи. Если же промышленный обмен данными идет поверх обычной локальной сети, то применяется полный стек протоколов, присущий конкретной локальной сети. Специфика обмена сосредоточена в этом случае только на седьмом (прикладном) уровне.

1.3. Специфика применения сетей для промышленной автоматизации

Область управления технологическими процессами сформировалась как отдельный раздел техники достаточно давно. За такими системами закрепилось исторически сложившееся название «Автоматизированная система управления технологическими процессами» (АСУ ТП). В настоящее время в связи с развитием цифровой техники данная система имеет иерархическую структуру (рис. 2).

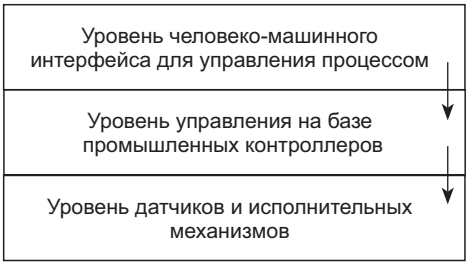


Рис. 2. Уровни иерархии системы АСУ ТП

Для функционирования системы все уровни должны иметь каналы связи как между уровнями, так и между объектами одного уровня. Таким образом, задача обмена информацией между агентами автоматизации является ключевой для успешного проектирования и внедрения систем АСУ ТП [5, с. 71]. Сети обмена данными в этом случае часто называются сетями промышленной автоматизации (СПА) или промышленными сетями. Можно выделить несколько функций СПА:

- передача информации от датчиков и исполнительных механизмов к управляющему контроллеру;
- обмен информацией между контроллерами, отвечающими за различные части технологического процесса;
- передача данных от контроллера на уровень человеко-машинного интерфейса (Human-Machine Interface — HMI).

Для каждой из вышеприведенных функций разработана своя группа сетевых решений, отличающаяся требованиями к обмену данными.

Критериями обмена данными являются:

- скорость обмена;
- максимальная протяженность сети;
- время получения информации (время цикла обмена);
- максимальное количество устройств, которое может быть подключено к сети.

Рассмотрим эти критерии более подробно.

Скорость обмена. В СПА, соединяющих ПЛК с нижним уровнем автоматизации, основное содержание передаваемой информации представляет собой показания датчиков и команды исполнительным механизмам. Этот объем информации невелик для средних систем автоматизации (содержащих до нескольких сотен датчиков), однако следует принимать во внимание требуемое время обновления показаний: быстроуправляемые системы (например, подвижные высокоскоростные объекты) могут требовать увеличения скорости обмена между контроллером и датчиками, что потребует увеличения скорости обмена по сети.

Например, типичными скоростями при обмене по специализированным сетям в АСУ ТП в области теплотехники и металлургии являются скорости 1–10 Мбит/с. СПА для пересылки данных на АРМ оператора (HMI-системы) более требовательна к скорости, т. к. объем информации, предоставляемый контроллером для системы верхнего уровня (SCADA-системы), как правило, включает в себя не только показания датчиков, но и информацию о состоянии технологического процесса (режимы работы, отсчитанные временные интервалы для прошедших или предстоящих событий, обработанные статистические данные и т. п). В связи с этим требования к скорости таких СПА приближаются к характеристикам современных локальных сетей: до 100 Мбит/с.

Максимальная протяженность сети. Так как промышленные сети, как правило, прокладываются в больших производственных помеще-

ниях (заводских цехах), они предъявляют повышенные требования к максимальным допустимым расстояниям между узлами. Установка контроллера зачастую невозможна в непосредственной близости от объекта управления, на котором локализовано подавляющее большинство датчиков и исполнительных механизмов. Следовательно, СПА должна поддерживать работу с длинными линиями передачи данных (до нескольких километров).

Время цикла обмена. Данный параметр определяется, исходя из следующих характеристик оборудования: скорости работы датчиков и ИМ, скорости работы контроллера, требований к системе управления объектом. Очевидно, что если датчик обеспечивает низкую скорость измерения параметра, использование более быстрого цикла его опроса нецелесообразно. В другой ситуации для быстроменяющихся характеристик объекта требуется оперативная передача данных к управляющему контроллеру, а значит и малое время цикла обмена. Однако, в любом случае, быстродействие управления будет ограничено скоростью работы контроллера. Для СПА, передающих данные на верхний уровень, время цикла обмена не так важно, как для управляющего процессом оборудования.

Максимальное количество устройств. Среди устройств, вступающих в сетевое взаимодействие, можно выделить следующие: управляющие контроллеры, агрегирующие показания датчиков вспомогательные контроллеры (удаленная периферия), интеллектуальные измерительные и исполнительные устройства, имеющие свой выход в сеть (частотные приводы, интегрированные узлы учета и т. п.), промышленные ПК и панели оператора. Общее количество устройств для больших АСУ ТП может достигать десятков и сотен. Таким образом, требуется, чтобы архитектура СПА позволяла подключиться и бесперебойно функционировать большому количеству устройств.

Контрольные вопросы

1. Каковы основные этапы развития технологии сетей передачи данных?
2. Какой международный стандарт описывает требования к открытой промышленной сети передачи данных?

3. Что называется моделью взаимодействия открытых систем (МВОС, OSI)?
4. Какую функцию выполняет каждый из уровней модели OSI?
5. В чем особенность сетей передачи данных в области промышленной автоматизации?
6. Какие характеристики обмена используются для оценки линии связи?
7. Какие скорости обмена являются типичными для сетей промышленной автоматизации?

2. Физические интерфейсы промышленных сетей

2.1. Характеристики линии связи

Для передачи информации в промышленности используются, как правило, цифровые линии связи: линии связи с конечным числом возможных состояний. Основные характеристики линии связи, имеющие значение для передачи цифровой информации:

- амплитудно-частотная характеристика;
- затухание;
- полоса пропускания;
- максимальная скорость передачи.

Амплитудно-частотная характеристика. При передаче через линию связи сигнал может быть представлен как набор гармонических колебаний разной частоты. Непрерывный или дискретный набор частот определяется из временной формы сигнала с помощью преобразования Фурье и называется спектр сигнала. В свою очередь, линия связи для каждой из частот частотного спектра имеет свое значения ослабления (затухания). Количественные значения ослабления для всех частот показывает амплитудно-частотная характеристика.

Затухание. Данный параметр показывает уменьшение амплитуды или мощности основной частоты спектра сигнала при передаче его по линии связи. Основная гармоника сигнала несет большую часть его мощности, и ее ослабление критически сказывается на способности линии пропускать такой сигнал. Затухание измеряется в децибелах (дБ). Так как затухание зависит от длины линии, то часто используется единица измерения в децибелах на единицу длины линии (на метр или на километр): дБ/км. Например, для кабеля, состоящего из витой пары, типичным является затухание –23 дБ на частоте 100 МГц при длине сегмента 100 м.

Полоса пропускания. Непрерывный диапазон частот, в пределах которого АЧХ линии связи не вызывает значительного искажения передаваемого сигнала, спектр которого лежит в данном диапазоне. Как правило, считается, что сигнал не искажается, если в пределах полосы пропускания лежит 90 % его мощности. Ширина полосы пропускания влияет на максимально возможную скорость передачи информации по линии связи.

Максимальная скорость передачи. Вышеназванные параметры характеризуют линию связи безотносительно способа кодирования передаваемой информации. Однако более интересен параметр, определяющий количество информации, которое можно передать по линии в единицу времени. Этот параметр называется максимальной скоростью передачи. Традиционно данный параметр для последовательных линий связи измеряется в битах в секунду. Различают техническую и информационную скорость. Техническая скорость измеряется в бодах в секунду и показывает скорость передачи, включающей не только полезную, но и все виды служебной информации (заголовки пакетов, контрольные суммы и т. д.). Информационная скорость показывает скорость «для пользователя», т. е. скорость передачи только полезной информации. Скорость передачи зависит не только от физических характеристик линии, но и от принятого способа физического кодирования, т. к. он определяет спектр передаваемого сигнала. Выбор оптимального способа кодирования позволяет добиться максимальной для данной линии связи скорости передачи с учетом ее физических характеристик.

Протоколы физического уровня модели OSI описывают различные методы физического кодирования информации.

2.2. Интерфейс последовательной передачи RS-232

Данный стандарт был введен в действие организацией EIA в 1960 г. под именем RS-232 (Recommended Standard) [7]. Устоявшаяся версия протокола под названием RS-232-C была утверждена стандартом в 1969 г. Первоначально протокол, описанный в стандарте, предназначался для подключения к аппаратуре, предназначенной для передачи данных (Data Terminal Equipment — DTE) устройств, отве-

чающих за непосредственное соединение с линией передачи (Data Communication Equipment — DCE) В качестве первого устройства, как правило, выступал компьютер, а в качестве второго — модем, соединенный с линией связи. В дальнейшем область применения протокола расширялась. Он использовался для подключения к компьютеру периферийных устройств прикладного назначения (принтер, сканер, мышь и др.), а также для связи компьютеров между собой. В настоящее время основная область его применения — связь со специализированными устройствами автоматизации (счетчики эл.энергии, расхода ресурсов и т. п.). Для связи по данному стандарту используется до десяти линий, протяженность которых не должна превышать 15 м. Стандарт также определяет тип разъема на каждом конце кабеля: D-образный 9- или 25-пиновый разъем типа male для DTE устройства и D-образный 9- или 25-пиновый разъем типа female для DCE-устройства. В современных ПК разъемы для последовательного интерфейса на корпусе системного блока отсутствуют. Как правило, они заменены на набор выводов прямо на материнской плате.

Назначение линий в соответствии со стандартом RS-232 приведено в табл. 2.

Таблица 2

Обозначение выводов стандарта RS-232

Обозначение	Расшифровка	Описание
PG	Protective Ground	защитное заземление (экран).
TxD	Transmitted Data	данные, передаваемые DTE
RxD	Received Data	данные, принимаемые DTE
RTS	Request To Send	сигнал запроса передачи к DCE
CTS	Clear To Send	сигнал готовности DCE к приему данных от DTE
DTR	Data Terminal Ready	готовность DTE к приему данных от DCE
DSR	Data Set Ready	Сигнал готовности DCE к приему и передаче данных
DCD	Data Carrier Detect	DCE принимает данные от удаленного DCE (обнаружена несущая)
RI	Ring	DCE принимает сигнал вызова из линии
SG	Signal Ground	сигнальное заземление, нулевой провод

Схема подключения DTE-устройства к DCE-устройству показана на рис. 3, *а*. Однако, несколько изменив порядок соединения линий, можно добиться соединения двух DTE-устройств, что позволяет образовать между ними канал передачи данных (рис. 3, *б*). Так как большая часть линий стандарта требуется для синхронизации работы DTE и DCE, то при соединении двух устройств DTE (двух компьютеров) эти линии остаются незадействованными. Для корректной работы микросхемы порта необходимо их замыкание на соответствующие парные выводы на ближнем конце (рис. 3, *в*). Таким образом, данное соединение требует кабеля, состоящего как минимум из 7 линий.

В настоящее время наиболее широко используется упрощенный способ соединения, в которой используется трехпроводная линия связи (линия приема, передачи и сигнальной земли) (рис. 3, *в*).

Так как интерфейс асинхронный, то посылки данных перемежаются промежутками простоя линии неопределенной длительности. Сигнал на линии в режиме простоя имеет уровень логического «1». Начало посылки данных отслеживается приемником по появлению на линии высокого уровня напряжения (логический «0»). Этот сигнал называется передачей стартового бита. За стартовым битом без разрывов идут информационные биты данных (от 5 до 8 бит). Затем передается необязательный бит контроля четности-нечетности. После него идет один или два стоповых бита, передаваемых уровнем логического «1». Затем линия возвращается в состояние покоя, либо начинается передача следующей посылки информации.

Интервал передачи одного бита определяется скоростью передачи по линии, и, в соответствии с стандартом RS-232 выбирается из ряда 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 бит/с. Важно, чтобы настройки скорости были одинаковыми на передающем и приемном конце. Кроме скорости к параметрам обмена относятся:

- количество информационных бит в посылке (от 5 до 8);
- наличие/отсутствие бита контроля четности/нечетности;
- количество стоповых бит (1 или 2).

Все эти параметры, включая скорость, должны быть заданы одинаковыми на обоих концах линии до начала сеанса приема-передачи по интерфейсу.

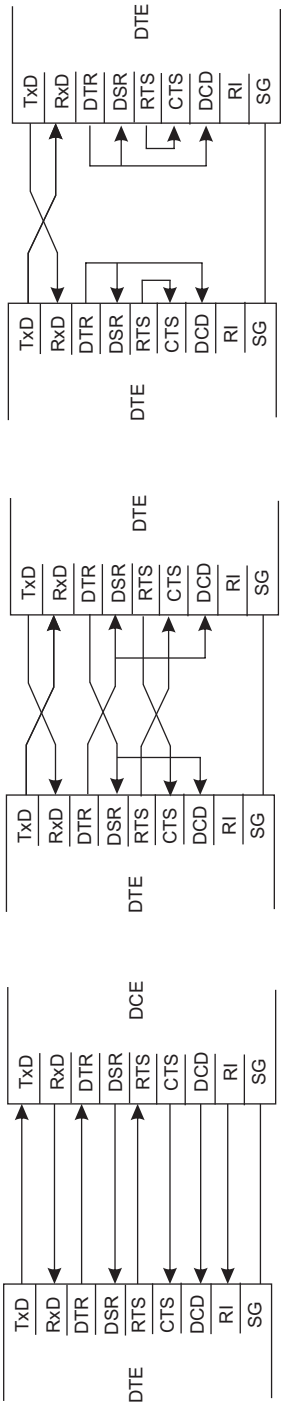


Рис. 3. Схемы подключения устройств по стандарту RS-232:
а) DTE-DCE; б) DTE-DTE (полный нуль-модемный кабель); в) DTE-DTE (упрощенный вариант)

2.3. Интерфейс последовательной передачи RS-485

Данный стандарт физического уровня для последовательного интерфейса был введен в действие в 1998 г. двумя организациями EIA и TIA [8]. В соответствии со стандартом RS-485 для передачи данных используются две линии, обычно обозначаемые как A и B. Опционально может присутствовать третья линия G для потенциала земли. Информация кодируется разностью потенциалов: $U_A - U_B < -200$ мВ — логическая единица, $U_A - U_B > 200$ мВ — логический ноль. Передача идет в последовательном режиме. Асинхронный символьный механизм передачи не описан в стандарте, но как правило состоит в том, что в состоянии покоя на линии находится сигнал, обозначающий логическую единицу, а о начале передачи одного символа говорит появление на линии логического нуля, за которым следуют 7 или 8 информационных битов. После информационных битов следует необязательный бит проверки на четность (или нечетность).

Для соединения нескольких устройств в одну общую сеть используется топология «общая шина», к которой может быть подключено до 32 устройств. Стандарт не оговаривает максимальной длины линии и максимальной скорости передачи. Однако, в процессе практического использования опытным путем было получено ограничение в скорости до 10 Мб/с и в расстоянии до 1,2 км (но не одновременно). Примерная эмпирическая формула говорит о том, что скорость в битах в секунду, умноженная на расстояние в метрах не должно превышать 10^8.

По своей сути RS-485 — это электрический интерфейс, поверх которого могут работать различные протоколы. В качестве физического интерфейса данный стандарт используют протоколы MODBUS, PROFIBUS.

2.4. Протоколы промышленных сетей на базе Ethernet-технологии

В настоящее время наибольшая часть используемого в локальных сетях оборудования и ПО соответствует группе стандартов Ethernet, описанной в спецификациях рабочей группы IEEE 802.3. Данные стандарты впервые были представлены в начале 80-х годов и содержали в себе сведения для физической организации локальной сети.

Текущая тенденция свидетельствует о все более широком использовании данного стандарта не только в локальных, но и в промышленных сетях. Это обусловлено несколькими его преимуществами:

- широкий выбор поддерживающего данный стандарт оборудования;
- открытость и доступность спецификаций, позволяющая проектировать и внедрять новые решения на его базе;
- большой объем имеющейся кабельной инфраструктуры, соответствующей данному стандарту;
- возможность подавать питание к устройствам по тем же кабелям, по которым идет поток данных.

В связи с особыми требованиями к передаче информации в АСУТП исходные спецификации Ethernet дополняются и дорабатываются производителями автоматизирующего оборудования в соответствии с их потребностями. Этот процесс может затрагивать основные моменты исходного стандарта (топологию, назначение линий и т. п.), что в свою очередь ведет к несовместимости выпускаемого оборудования с исходным стандартом от группы IEEE 802.3.

Можно выделить несколько способов использования этой спецификации в оборудовании АСУТП:

- **использование без модификации**. В этом случае уровни МВОС с первого по четвертый сохраняются неизменными. Протоколы прикладного уровня определяются функциональным назначением оборудования и/или ПО. У такого подхода есть как плюсы, так и минусы. Основной плюс: производимый продукт полностью совместим со стандартом IEEE 802.3 и может использовать существующую инфраструктуру локальных сетей для передачи данных между автоматизирующими устройствами. Например, можно подключить к существующей локальной сети ПЛК и ПК с запущенной на нем SCADA-системой. Если оба устройства поддерживают существующий стандарт Ethernet, то поток данных между ними будет корректно проходить через локальную сеть. Примерами протоколов, относящихся к этой группе, являются MODBUS TCP, EtherNet/IP.
- **использование без изменений физического уровня с модификацией вышележащих уровней**. Стандарт Ethernet описывает первый (физический) и второй (канальный) уровни модели МВОС. Однако, зачастую удобнее заменить канальный уровень на специ-

фический протокол, наилучшим образом соответствующий для решения задач автоматизации. Например, исходный стандарт никак не регламентирует и не гарантирует время доставки пакета с данными. Однако путем детерминированности доступа к среде передачи (которой нет в исходном стандарте) можно добиться гарантированного времени доставки и отклика, что является важным в условиях управления техническими системами. Примером протокола, относящегося к этой группе является PROFINET RT/IRT.

- **модификация всех уровней, включая физический**. Данный вариант использования делает устройства несовместимыми со спецификациями IEEE 802.3, однако позволяет наиболее эффективно использовать возможности линий связи путем модификации как уровней сигналов, так и механизма доступа к среде. Примером протокола, относящегося к этой группе является EtherCAT.

Наиболее часто в качестве физического уровня используется кабель типа «витая пара» категории 5 или выше. Спецификация характеристик кабеля определена стандартом TIA/EIA-568-B от 2001 г [9]. Таблица 3 показывает затухание в сегменте кабеля длиной 100 м на различных частотах передачи при 20 °C.

Таблица 3

Зависимость частоты и затухания для различных категорий кабеля UTP [9]

Частота, МГц	Затухание, дБ	
–	Cat 3	Cat5e
1	4,2	2,2
4	7,3	4,5
8	10,2	6,3
10	11,5	7,1
16	14,9	9,1
20	–	10,2
25	–	11,4
31,25	–	12,9
62,5	–	18,6
100	–	24,0

Стандарт определяет только важнейшие характеристики кабеля (затухание, наводки на ближнем конце, и некоторые другие). Остальные характеристики выпускаемого кабеля зависят от производите-

ля. Для примера в табл. 4 приведены типичные характеристики этого типа кабеля.

Таблица 4

Типичные характеристики кабеля типа «витая пара»

Характеристика	Номинальное значение
Волновое сопротивление при 100 МГц	100±15 Ом
Номинальный импеданс при 100 МГц	100±5 Ом
Фазовая задержка	4,80–5,30 нс/м
Индуктивность	525 нГ/м
Частота среза	⩽57 кГц
Диаметр жилы	AWG-24 (0,51054 мм)
Толщина изоляции	0,245 мм
Максимальный ток в проводнике	0,577 А
Рабочая температура	От –55 до +60 °С
Максимальное рабочее напряжение	125 V (DC)

К преимуществам кабеля типа «витая пара» относятся следующие:

- простота монтажа кабельной инфраструктуры, которая аналогична используемой для низковольтных сетей подачи питания;
- простота используемого оконечного оборудования для приема и передачи данных.

К недостаткам данного типа относятся:

- сильное затухание в кабеле, не позволяющее строить протяженные сети без использования активных повторителей сигнала;
- подверженность электромагнитным помехам.

Для протяженных линий используется волоконно-оптический кабель. Существует несколько стандартов данного кабеля. Наиболее часто используемые были разработаны организацией ITU-T и относятся к группе «Волоконно-оптические кабели» под номерами G.650–G.659. Как пример можно привести характеристики наиболее широко применяемого одномодового кабеля с несмещенной дисперсией, описанные в документе G.652 [10] (табл. 5).

Использование волоконно-оптического кабеля позволяет добиться следующих преимуществ:

- протяженность линий связи возрастает до единиц и десятков километров;

- вследствие того, что кабель сделан из непроводящего ток материла, возможно прокладывание линий вне зданий, не опасаясь повреждения их атмосферным электричеством;
- малое затухание в широкой полосе пропускания позволяет повысить информационную пропускную способность линии путем частотного уплотнения каналов (технология WDM).

К недостаткам данного типа кабеля относятся:

- сложность монтажа и ремонта провода, требующие наличия специального оборудования (сварочного аппарата для оптики);
- повышенные требования к геометрии кабеля при прокладке, вызванные сложным характером распространения световой волны в кабеле.

Таблица 5

Характеристики оптического кабеля в соответствии со стандартом G.652

Характеристика	Параметр характеристики	Значение	Единицы измерения
Диаметр модового поля	Длина волны	1310	нм
	Диапазон номинальных значений	8,6–9,5	мкм
	Допуск	±0,6	мкм
Диаметр оболочки	Номинал	125,0	мкм
	Допуск	±1	мкм
Погрешность концентричности сердцевины	Максимальное значение	0,6	мкм
Некруглость оболочки	Максимальное значение	1,0	%
Кабельная длина волны отсечки	Максимальное значение	1260	нм
Потери на макроизгибе	Радиус	30	мм
	Число витков	100	–
	Максимум на 1635 нм	0,1	дБ
Предел прочности	Минимум	0,69	ГПа
Параметр хроматической дисперсии	λ_{0max}	1300	нм
	λ_{0max}	1324	нм
	λ_{0max}	0,092	пс/(нм^2 × км)
Коэффициент затухания	Максимум на 1310 нм	0,4	дБ
	Максимум на 1550 нм	0,35	дБ
	Максимум на 1625 нм	0,4	дБ

Контрольные вопросы

1. Какие характеристики линии связи используются для оценки ее качества?
2. Сколько линий используется для передачи по интерфейсу RS-232?
3. Какими уровнями разности потенциалов кодируется 0 и 1 при передаче по интерфейсу RS-485?
4. Какой стандарт описывает передачу данных по технологии Ethernet?
5. Какие характеристики передачи имеет линия, выполненная по технологии «витая пара»?
6. Какая максимальная дальность передачи может быть достигнута при использовании волоконно-оптического кабеля?
7. В чем заключаются преимущества и недостатки использования волоконно-оптической связи?

3. Протокол MODBUS

3.1. Протокол MODBUS RTU

Протокол MODBUS RTU [11] (Remote Terminal Unit) был разработан компанией Modicon в 1979 г. для использования в ПЛК данного производителя. В настоящее время данный протокол является «де-факто» стандартом, используемым для связи между собой устройств в промышленной сети.

В качестве физического уровня данный протокол использует последовательные интерфейсы RS-232 и RS-485. На канальном уровне предусмотрены специфический для протокола формат кадров ADU (Application Data Unit). Прикладной уровень описывается форматом кадра PDU (Protocol Data Unit).

Протокол описывает детерминированный механизм доступа к среде по типу «ведущий-ведомый». Детерминированность в такого типа обмене достигается тем, что в каждый момент времени активность в среде передачи определяется одним ведущим устройством: оно формирует адресные запросы к ведомым устройствам и инициирует их на ответные действия (формирование пакета с ответными данными). В сети MODBUS ведущим может выступать только одно устройство из всех присутствующих в любой момент времени.

Участниками взаимодействия по протоколу являются два типа устройства: ведущее (master) и ведомое (slave). Ведущее устройство может формировать запросы к ведомому устройству. Ведомое устройство отвечает на запросы ведущего. Инициатором обмена всегда выступает ведущее устройство.

В протоколе описаны следующие типы данных:

Discrete Inputs (DI) — дискретные входы: однобитовый тип, доступен только для чтения.

Coils (CI) — однобитовый тип, доступен для чтения и записи.

Input Registers (IR) — регистры входов: 16-битовый знаковый или беззнаковый тип, доступен только для чтения.

Holding Registers (HR) — регистры хранения: 16-битовый знаковый или беззнаковый тип, доступен для чтения и записи.

Взаимодействие состоит в получении ведущим устройством данных вышеуказанного формата, расположенных в памяти ведомого устройства, либо в записи в память ведомого информации, переданной ему ведущим. Для всех типов взаимодействия используется кадр PDU следующего формата:

код функции	данные
1 байт	<253 байта

Код функции кодируется однобайтовым полем и может принимать значения в диапазоне 1...127. Диапазон значений 128...255 зарезервирован для кодов ошибок.

В качестве примера рассмотрим кадр PDU-запроса регистров хранения протокола MODBUS:

[0x03][A1][A0][Q1][Q0],

здесь [0x03] — байт кода запроса из области регистров хранения; [A1] [A0] — старший и младший байты адреса первого запрашиваемого регистра; [Q1] [Q0] — старший и младший байты количества запрашиваемых подряд идущих регистров.

Кадр PDU-ответа ведомого устройства содержит информацию

[0x03] [N] [D],

где [N] — байт, содержащий длину передаваемых данных; [D] — N байт данных.

Количество данных в кадре ответа ограничено в стандарте 253 байтами или 126 16-битовыми регистрами.

Для передачи пакета по физическим линиям связи кадр PDU помещается в поле данных кадра канального уровня ADU. Формат кадра ADU зависит от типа среды передачи.

Пример формата кадра ADU для передачи по интерфейсу RS-485

Адрес ведомого устройства	Кадр PDU	Блок обнаружения ошибок
1 байт	<=253 байта	2 байта

3.2. Протокол MODBUS TCP

В последнее время в связи с распространением инфраструктуры локальных и глобальных сетей стала использоваться модификация протокола MODBUS, использующая в качестве среды передачи сеть на базе технологии Ethernet. Данная модификация называется MODBUS TCP. В основе обмена по данному протоколу лежит использование существующего в локальных сетях стека протоколов TCP/IP, поверх которого передаются пакеты ADU определенного спецификацией MODBUS TCP-формата.

Формат кадра ADU для передачи поверх TCP/IP

ID-транзакции	ID-протокола	Длина пакета	Адрес ведомого устройства	Кадр PDU
2 байта	2 байта	2 байта	1 байт	<=253 байта

Где ID-транзакции — служит для синхронизации сообщений между сервером и клиентом, обычно не используется (заполняется нулями); ID-протокола — 0 для MODBUS TCP; длина пакета — старший и младший байты длины следующей за этим полем части пакета; адрес ведомого устройства — адрес подчиненного устройства, к которому адресован запрос. Обычно игнорируется, если соединение уже установлено с конкретным устройством, или в системе только одно устройство. Может использоваться, если соединение установлено с мостом, который связан физически, например, с сетью RS-485.

Использование протокола MODBUS TCP имеет особенности, которые важно знать.

Поскольку сетевая инфраструктура составляет сеть на базе протокола IP, требуется присвоить каждому из узлов (и ведущему и ведомым) IP-адреса из существующей сети. Адрес ведомого устройства должен быть известен ведущему устройству для формирования запросов к нему.

В качестве транспорта используется протокол TCP, предполагающий установление соединения перед обменом полезными данными. Для установления соединения необходимо конфигурирование ролей партнеров. В частности, ведомое устройство должно быть сконфигурировано как TCP-сервер с открытым в сеть TCP-портом. Сервер пассивно ожидает прихода пакетов на сконфигурированный порт для установления соединения и последующего обмена данными.

Ведущее устройство должно быть сконфигурировано как TCP-клиент. В начале работы оно устанавливает соединение с TCP-сервером по заранее известным IP-адресу сервера и TCP-порту. Затем циклически начинает опрашивать ведомое устройство, передавая ему ADU-кадры запросов, получая ответы и выводя информацию из них на экран.

3.2.1. Организация доступа к серверу посредством работы с сокетами

Для обмена данными по протоколу MODBUS TCP предварительно необходимо установить TCP-соединение между клиентом и сервером. Оконечными точками для установления соединения являются IP-адреса узлов и TCP порты процессов, запущенных на этих узлах. IP-адрес представляет собой 32-битный адрес (для протокола IPv4, 128-битный для IPv6), наиболее часто его представляют в символьной форме mmm.nnn.ppp.qqq (адрес, разбитый на четыре поля, разделённых точками, по одному байту в поле). TCP-порт — это номер порта в диапазоне от 0 до 65535.

Эта пара и есть сокет («гнездо», соответствующее адресу и порту). Например, при обращении к серверу с IP-адресов 192.168.1.1 на HTTP-порт сокет будет выглядеть так: 192.168.1.1:80.

В процессе обмена, как правило, используется два сокета — сокет отправителя и сокет получателя. Процесс-сервер может создать «слушающий» сокет (серверный сокет) и привязать его к какому-нибудь порту. Процесс-клиент создает клиентский сокет и подсоединяет его к серверному сокету.

Работа с сокетами в ОС Windows включает в себя:

- создание сокета;
- связывание сокета (для серверных сокетов);
- ожидание установления соединения (для серверных сокетов);

- запрос на установление соединения (для клиентских сокетов);
- функции обмена данными.

Алгоритм работы с клиентским сокетом имеет вид:

1. Создание и инициализация сокета;
2. Запрос на установление соединения с серверным сокетом на удаленном узле;
3. Если запрос принят, то установка соединения и обмен данными (запросы и ответы), иначе — завершение программы;
4. Завершение соединения.

Рассмотрим функции языка C++ для работы с сокетами.

Для работы с сокетами необходимо добавить ссылку на заголовочный файл с соответствующими функциями

```
#include <winsock2.h>
```

Также перед использованием сокетов нужно инициализировать систему работы с сокетами, вызвав функцию WSAStartup:

```
char buff [1024];
if (WSAStartup (0x0202, (WSADATA *)&buff [0]))
{
//здесь выполняем действия в случае неуспешной инициализации
}
```

После инициализации можно создавать сокет. Создание сокета включает в себя вызов функции

```
SOCKET sock=socket (AF_INET, SOCK_STREAM, 0);
```

которой передаются указания на схему адресации (для Ethernet сетей используется константа AF_INET), режим создаваемого сокета (в работе используется сокет с установлением соединения, константа для которого имеет вид SOCK_STREAM) и тип протокола (в нашем случае 0 для протокола TCP).

При успешном создании сокета функция возвращает идентификатор сокета — переменную типа SOCKET. Этот идентификатор используется в дальнейшем при вызове других функций. В случае неудачного завершения данная функция возвращает значение INVALID_SOCKET.

После инициализации выполняется попытка подключения к удаленному сокету с помощью вызова функции connect, которой передается идентификатор сокета, полученный от функции socket, указа-

тель на структуру sockaddr_in, содержащую информацию о удаленном сокете. Инициализация сокета и вызов функции соединения показаны в фрагменте кода:

```
SOCKET sock;
sockaddr_in addr;

sock = socket (AF_INET, SOCK_STREAM, 0);
addr.sin_family = AF_INET;
addr.sin_port = htons (2000);//выбираем порт подключения
addr.sin_addr.s_addr = inet_addr ("192.168.1.1");
//выбираем адрес подключения
if (connect (sock, (sockaddr *)&addr, sizeof (addr)) == SOCKET_ERROR)
{
//здесь выполняем действия, требуемые в случае
//неуспешного установления соединения
}
else
{
//здесь выполняем действия, требуемые в случае
//успешного установления соединения
}
```

В случае успешного установления соединения можно принимать и передавать данные с помощью функций send и recv. Перед вызовом метода send необходимо сформировать массив байтов для передачи. Например, если требуется передать 100 байт удаленному партнеру, расположенных в массиве buffer, вызов метода send будет иметь вид

```
char buffer [100];
int NumBytes = 100;
//здесь заполняем массив buffer
if (send (sock, buffer, NumBytes, 0) == SOCKET_ERROR)
{
//здесь выполняем действия, требуемые в случае
//неуспешной передачи
}
```

Метод работает в синхронном режиме, что означает что пока данные не будут отправлены в сеть или не будет получена какая-либо ошибка передачи, программа не продолжит выполнение дальше его вызова.

Аналогично выполняется вызов функции получения данных

```
recv(sock, buffer, NumBytes, 0);
```

В этом случае количество фактически считанных байт будет возвращено функцией recv, а сами данные помещены в массив buffer. Метод также работает в синхронном режиме.

Для обработки исключений нужно анализировать возвращаемое функциями значение, сравнивая его с константой SOCKET_ERROR. Детальную информацию об ошибке можно получить, вызвав функцию WSAGetLastError непосредственно после этого.

При завершении работы с сокетами необходимо закрыть соединение вызовом функции closesocket, которой передается идентификатор сокета.

```
closesocket(sock);
```

Функции языка C# для работы с сокетами

Для работы с сокетами необходимо добавить в проект ссылки на библиотеки System.Net и System.Net.Sockets, а также добавить в файл с кодом использование их пространств имен:

```
using System.Net;
using System.Net.Sockets;
```

Создание сокета включает в себя создание объекта класса Socket, конструктору которого передается указание на схему адресации (для Ethernet сетей используется константа AddressFamily.InterNetwork), режим создаваемого сокета (в работе используется потоковый сокет, константа для которого имеет вид SocketType.Stream) и тип протокола (в нашем случае ProtocolType.Tcp).

```
Socket ConnSoc;
ConnSoc = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
```

После инициализации выполняется попытка подключения к удаленному сокету с помощью вызова метода Connect, которому передается ссылка на объект класса IPEndPoint, содержащий информацию о удаленном сокете. Инициализация данного класса и вызов функции соединения показаны в фрагменте кода

```
byte[] IpAddr=new byte[4];
ushort Port;
```

```
   IPAddress address;
   IPEndPoint ipe;
   Socket ConnSoc;

   ConnSoc = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
   IpAddr [0]=192;
   IpAddr [0]=168;
   IpAddr [0]=1;
   IpAddr [0]=1;
   Port=2000;
   address = new IPAddress(IpAddr);
   ipe = new IPEndPoint(address, Port);
   ConnSoc.Connect (ipe);
   ConnSoc.ReceiveTimeout = 30000;
   ConnSoc.SendTimeout = 30000;
   if (! ConnSoc.Connected)
   {
   //здесь выполняем действия, требуемые в случае
   //неуспешного установления соединения
   }
   else
   {
   //здесь выполняем действия, требуемые в случае
   //успешного установления соединения
   }
```

В случае успешного установления соединения можно принимать и передавать данные с помощью методов Send и Receive.

Перед вызовом метода Send необходимо сформировать массив байтов для передачи. Например, если требуется передать 100 байт удаленному партнеру, расположенных в массиве buffer, вызов метода Send будет иметь вид

```
   byte[] buffer=new byte[100];
   int NumBytes=100;
   //здесь заполняем массив buffer
   ConnSock.Send(buffer, NumBytes, SocketFlags.None);
```

Метод работает в синхронном режиме, что означает что пока данные не будут отправлены в сеть или не будет получена какая-либо ошибка передачи, программа не продолжит выполнение дальше его вызова. Для предотвращения зависания перед обменом данными устанавливаются значения переменных SendTimeout, ReceiveTimeout, которые определяют максимальное отведенное время для приема или передачи, измеряемое в мс.

Аналогично выполняется вызов функции получения данных

```
ConnSoc.Receive(buffer, Length, SocketFlags.None);
```

В этом случае количество фактически считанных байт будет помещено в переменную Length а сами данные — в массив buffer.

Метод также работает в синхронном режиме.

Для обработки исключений при работе с сокетами необходимо все операции с ними поместить в блок try..catch вида

```
try
{
//здесь работаем с сокетами
}
catch (SocketException e)
{
MessageBox.Show("Error!", String.Format (" {0} Sending error code: {1}.", e.Message, e.ErrorCode), MessageBoxButtons.OKCancel, MessageBoxIcon.Asterisk);
}
```

При завершении работы с сокетами необходимо закрыть соединение вызовом метода Disconnect.

```
ConnSoc.Disconnect(false);
```

Контрольные вопросы

1. Какие роли предусматривает для устройств-участников протокол MODBUS?
2. Какие типы данных описаны протоколом MODBUS?

3. К каким уровням модели OSI принадлежат кадры ADU и PDU, описанные в протоколе MODBUS?
4. Какое максимальное количество байт данных может быть передано одним кадром PDU?
5. Какие особенности имеются у протокола MODBUS TCP?
6. Что называется «сокетом» при передаче данных с помощью протокола MODBUS TCP?
7. Каков алгоритм работы с клиентским сокетом для организации обмена по протоколу MODBUS TCP?

4. Протокол PROFIBUS

4.1. Обзор стандарта

Стандарт PROFIBUS появился как ответ на требования стандартизации промышленных сетевых решений в Германии в первой половине 80-х гг. XX в. [12]. Значительное влияние на его формирование было оказано специалистами международного концерна Siemens AG (подразделения, занимающегося промышленной автоматизацией). В настоящее время данный концерн является основным разработчиком вариаций данной платформы. Несмотря на то, что данный стандарт декларирован как открытый, открытыми являются только нижние уровни сетевого взаимодействия. Протоколы доступа к среде, форматы кадров, интерфейсы взаимодействия с приложениями прикладного уровня регламентируются проприетарными протоколами фирмы Siemens. Для получения доступа к ним требуется вступление в группу разработчиков стандарта PROFIBUS.

4.2. Физический уровень

Физический уровень включает в себя определение длины линии, топологию, интерфейсы подключения, количество устройств и скорость передачи (в диапазоне от 9,6 до 1500 кбит/с).

Характеристики, определенные в первой версии стандарта (соответствуют стандарту EIA RS-485), приведены в табл. 6.

Сегменты сети можно соединять между собой активными повторителями. При этом допускается присутствие не более трех повторителей между двумя станциями.

Таблица 6

Характеристики линии при передаче по протоколу RS-485

Параметр	Значение
Топология	Общая шина с оконечными сопротивлениями на обоих концах
Среда передачи	Экранированная витая пара
Длина линии	<=1200 м
Количество устройств	32 (ведущие, ведомые, повторители)
Скорости передачи	9,6/19,2/93,75/187,5/500/1500 кбит/с
Отказоустойчивость	Возможно использование второй среды передачи
Адресация	Адреса от 0 до 127 (127 используется для широковещательной рассылки)
Типы устройств	Ведущее (master) — активное устройство, контролирующее шину Ведомое (slave) — пассивное устройство без контроля шины
Доступ к шине	Смешанный (централизованно-децентрализованный). Передача маркера между ведущими станциями и ведущий-ведомый обмены между ведущим и ведомыми
Длина кадра	От 1–3 до 255 байт

Во второй версии стандарта были повышены требования по искробезопасности, что вызвало следующие изменения: в качестве среды передачи, кроме витой пары, стало возможно использовать многожильные кабели (экранированные или нет), максимальная длина увеличена до 1900 м, введена новая скорость 31,25 кбит/с.

Для подсоединения устройства к шине используется 9-типиновый D-образный коннектор.

Таблица 7

Назначения линий разъема для подключения к интерфейсу

Номер пина	RS-485	Название	Назначение
1		SHIELD**	Экран, защитная земля
2		M24V**	Минус 24 В выходного напряжения
3	B/B	RxD/TxD-P	Прием-передача данных (проводник B)
4		CNTR-P**	Управление (P-линия)
5	C/C	DGND	Сигнальная земля
6		VP*	Положительное напряжение
7		P24V**	Плюс 24 В выходного напряжения
8	A/A	RxD/TxD-N	Прием-передача данных (проводник A)
9		CNTR-N**	Управление (N-линия)

Примечание: * — линия необходима только для станции на конце линии;
** — необязательная линия

Управляющие линии могут использоваться для передачи управляющих сигналов между устройствами. Линии питания (пины 2 и 7) могут использоваться для питания устройств посредством сети PROFIBUS (например, если устройства не имеют встроенного блока питания). При этом устройство в соответствии со стандартом IEC 1131–2 не должно потреблять более 100 мА.

Спецификацией определены два типа кабеля, незначительно отличающиеся механическими и электрическими характеристиками (A и B).

Таблица 8

Спецификация кабеля для передачи по протоколу RS-485

Параметр	Тип A	Тип B
Волновое сопротивление	135–165 Ом (f = 3–20 МГц)	100–130 Ом (f > 100 кГц)
Емкость	< 30 пФ/м	< 60 пФ/м
Сопротивление	< 110 Ом/км	-
Площадь сечения	>= 0,34 мм^2	>= 0,22 мм^2

Скорость передачи связана с дальностью обратной зависимостью.

Таблица 9

Зависимость предельной скорости передачи от длины сегмента

Бодовая скорость, кбит/с	Длина, м (кабель типа A)	Длина, м (кабель типа B)
9,6	1200	1200
19,2	1200	1200
93,75	1200	1200
187,5	1000	600
500	400	200
1500	200	70

В качестве способа кодирования для протокола PROFIBUS-DP был выбран NRZ-код, при котором логический нуль кодируется отрицательной разностью потенциалов, а логическая единица — положительной разностью потенциалов. Для передачи символов используется асинхронный посимвольный способ передачи (UART). Представление бит при передаче одного символа (октета из 8 бит) представлено на рис. 4.

Рис. 4. Схема передачи UART-символа

В состоянии покоя на линии присутствует сигнал логической единицы. Для синхронизации приемника посылка начинается с стартбита (бита логического нуля), который изменяет состояние линии. За ним идут 8 бит данных, за которыми следует бит четности/нечетности. Данный бит позволяет обнаружить ошибки, возникающие при передаче. Завершается передача символа стоп-битом, значение которого равно 1. После него линия возвращается в состояние покоя, либо начинается передача следующего символа.

Кроме экранированной витой пары, PROFIBUS предусматривает возможность использования в качестве среды передачи оптоволоконного кабеля. Для оптической передачи данных в сетях PROFIBUS используются либо встроенные оптические порты оптических шинных терминалов (OBT), либо модули оптической связи (OLM). В кабеле требуется наличие двух волокон. Модули со встроенными оптическими портами и оптическими шинными терминалами (OBT) можно соединять между собой только по топологии «общая шина». Модули OLM можно соединять в топологии «общая шина», «звезда» и «кольцо».

Таблица 10

**Характеристики сети PROFIBUS,
построенной на оптоволоконной среде передачи данных**

Параметр	Показатели
Топология сети	«Общая шина» для встроенных оптических портов и терминалов OBT; «Общая шина», «Звезда», «Кольцо» при использовании OLM
Среда передачи	Волоконно-оптические кабели с волокнами из стекла, с пластиковым покрытием или с пластиковыми волокнами
Длины сегментов (точка-точка)	До 15 км при использовании стекловолокна в зависимости от типа волокна и OLM: при использовании пластиковых волокон, м OLM: 0–80; OBT: 1–50

Окончание табл. 10

Параметр	Показатели
Скорость передачи	9,6 кбит/с, 19,2 кбит/с, 45,45 кбит/с, 93,75 кбит/с, 187,5 кбит/с, 500 кбит/с, 1,5 Мбит/с, 3 Мбит/с*, 6 Мбит/с*, 12 Мбит/с
Количество узлов	Не более 127 в сети (126 для кольцевой топологии с использованием OLM)

* Кроме случаев использования оптических портов и ОВТ.

Также существуют технологии беспроводной передачи данных по сети PROFIBUS с использованием инфракрасного сигнала. Для обмена данными требуется прямая видимость между узлами. Максимальное расстояние передачи равно 15 м. Беспроводные сети строятся с помощью модулей инфракрасной связи (ILM). Подключение проводного сегмента выполняется к порту модуля ILM.

Таблица 11

Характеристики сети PROFIBUS, построенной на инфракрасной среде передачи данных

Параметр	Показатели
Топология сети	«Точка» — «точка» «Точка» — «многоточка»
Среда передачи	Открытое пространство с прямой видимостью
Максимальная длина сегмента	15 м
Скорость передачи модуля ILM	9,6 кбит/с, 19,2 кбит/с, 45,45 кбит/с, 93,7545 кбит/с, 187,5 кбит/с, 500 кбит/с, 1.5 Мбит/с
Количество узлов	Не более 127 в сети

4.3. Канальный уровень

Протокол доступа к среде определен в соответствии со стандартом DIN 19 241–2, IEC 955 (PROWAY C), ISO 8802–2 и ISO/IEC JTC 1/SC 6N 4960 (LLC-тип 1 и 3).

Передача по протоколу UART использует асинхронную передачу восьми бит данных с старт-стопными битами, как определено в IEC 870-5-1.

Определены следующие сервисы для передачи данных:

- Посылка данных с подтверждением (SDA). Позволяет передать данные одиночному удаленному устройству и получить подтверждение о передаче или ошибке передачи;

- Посылка данных без подтверждения (SDN). Позволяет передать данные одному или нескольким (multicast) или всем (broadcast) устройствам одновременно, без подтверждения доставки;
- Посылка и запрос данных с ответом (SRD). Позволяет передать данные одному устройству, одновременно запрашивая данные с этого устройства. Если произошла ошибка, передача данных повторяется;
- Циклическая посылка и запрос данных с ответом (CSRD). Позволяет непрерывно передавать данные одному устройству, одновременно запрашивая данные с этого устройства.

Каждый из сервисов определяется как совокупность и последовательность обмена элементарными кадрами (frame).

Посылка данных с подтверждением инициируется на канальном уровне локального ведущего устройства для передачи данных в виде кадра данных (link service data unit) на удаленное ведущее или ведомое устройство. На удаленном устройстве, если кадр был принят без ошибок, он передается канальным уровнем на вышележащие уровни. Локальное устройство получает подтверждение о доставке или недоставке данных от удаленного устройства. Если при передаче произошли ошибки, канальный уровень локального устройства должен повторить передачу данных.

Посылка данных без подтверждения позволяет локальному устройству передать данные на одно, несколько или все удаленные устройства. При этом на локальном устройстве имеется только подтверждение отправки, но не подтверждение доставки данных. Если на удаленном устройстве кадр был принят без ошибок, он передается канальным уровнем на вышележащие уровни.

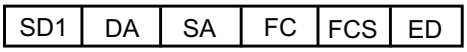
Посылка и запрос данных с ответом позволяют локальному устройству переслать данные на удаленное устройство и одновременно этим же кадром запросить с него данные. Если данных для передачи нет, кадр данных может содержать только запрос. Локальное устройство получает или запрошенные данные, или подтверждение того, что данные недоступны на удаленном устройстве, или подтверждение о недоставлении данных на удаленное устройство.

Циклическая посылка и запрос данных с ответом позволяют ведущему устройству циклически передавать данные на удаленное устройство. Если на удаленном устройстве кадр был принят без ошибок, он передается канальным уровнем на вышележащие уровни. Сервис так-

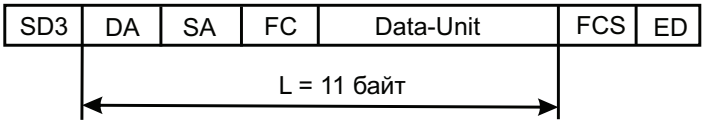
же позволяет циклически запрашивать данные с удаленного устройства и получать их. Список удаленных устройств и последовательность передачи данных и запросов должны быть определены на локальном устройстве в виде списка опроса (poll list).

Форматы кадров

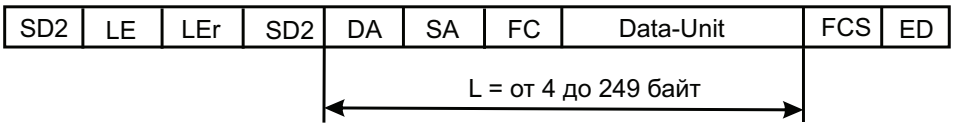
1. Формат с постоянной длиной информационного поля



2. Формат с постоянной длиной информационного блока с данными



3. Формат с переменной длиной информационного блока



4. Короткое квитирование

5. Телеграмма-токен (маркер)

Здесь L — длина информационного поля;

SC — Single Character — отдельный символ, используемый только для подтверждения (SC=E5h);

SD1–SD4 — Start Delimiter — начальный ограничитель кадра (SD1=10h, SD2 = 68h, SD3 = A2h, SD4 = DCh);

LE/LEr — Length — указывает длину информационных полей у кадров переменной длины;

DA — Destination Address — PROFIBUS-адрес приемника;

SA — Source Address — PROFIBUS-адрес устройства, сгенерировавшего кадр;

FC — Frame Control — байт содержит в себе информацию о службе данного сообщения и о приоритете сообщения;

Data-Unit — поле данных;

FCS — Frame check Sequence — контрольная сумма кадра (побайтно применяет к байтам кадра операцию И);

ED — End Delimiter — ограничитель конца кадра (ED = 16h).

Каждый кадр должен предваряться сигналом простоя линии (лог. 1) в течение не менее чем 33 битовых интервала. Между передачей двух последовательных символов в кадре не должно быть промежутков времени (символы передаются последовательно).

Доступ к среде передачи позволяет решить две взаимоисключающие задачи:

- каждому из устройств необходимо обеспечить гарантированный доступ к общей среде передачи в течение определенного временного промежутка заранее известной длительности;
- для оперативного управления технологическим процессом требуется максимально быстрый обмен данными между управляющим контроллером и простой полевой периферией (датчиками и исполнительными механизмами).

Эти задачи решаются с помощью децентрализованной передачи маркера между ведущими устройствами в сети и централизованным опросом ведомых устройств в случае, когда их ведущее устройство получает маркер и становится активным на шине.

Активное ведущее устройство, которое владеет в данный момент маркером, монопольно использует среду передачи для опроса ведомых устройств по списку опроса. Когда опрос закончен, ведущее устройство выдает на шину кадр с маркером, в котором указан адрес следующего ведущего устройства, и, таким образом, передает монопольный доступ к среде передачи данному устройству.

Ведущие устройства упорядочены по возрастанию их адресов в логическое маркерное кольцо и передают друг другу маркер владения шиной последовательно (рис. 5).

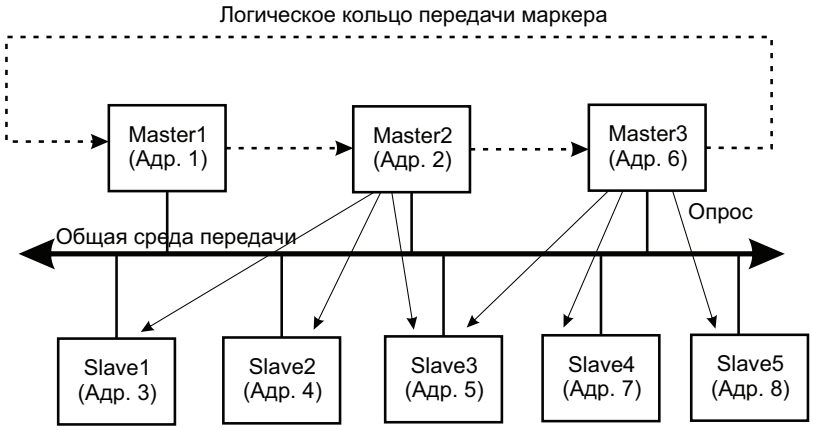


Рис. 5. Схема взаимодействия устройств по протоколу PROFIBUS

Время одного обращения маркера между всеми ведущими устройствами называется временем обращения маркера и ограничивается заданным максимальным значением данной величины.

Для примера на рис. 6 и 7 приведены характеристики сети PROFIBUS, состоящей из трех устройств (ПЛК Siemens S7–300, операторская панель, ПК с SCADA-системой для визуализации).

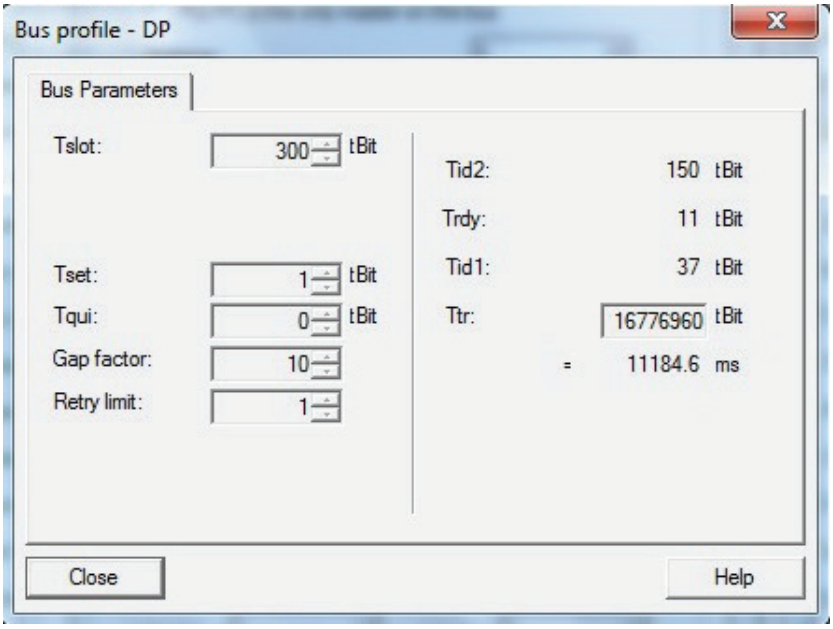


Рис. 6. Временные характеристики сегмента сети PROFIBUS

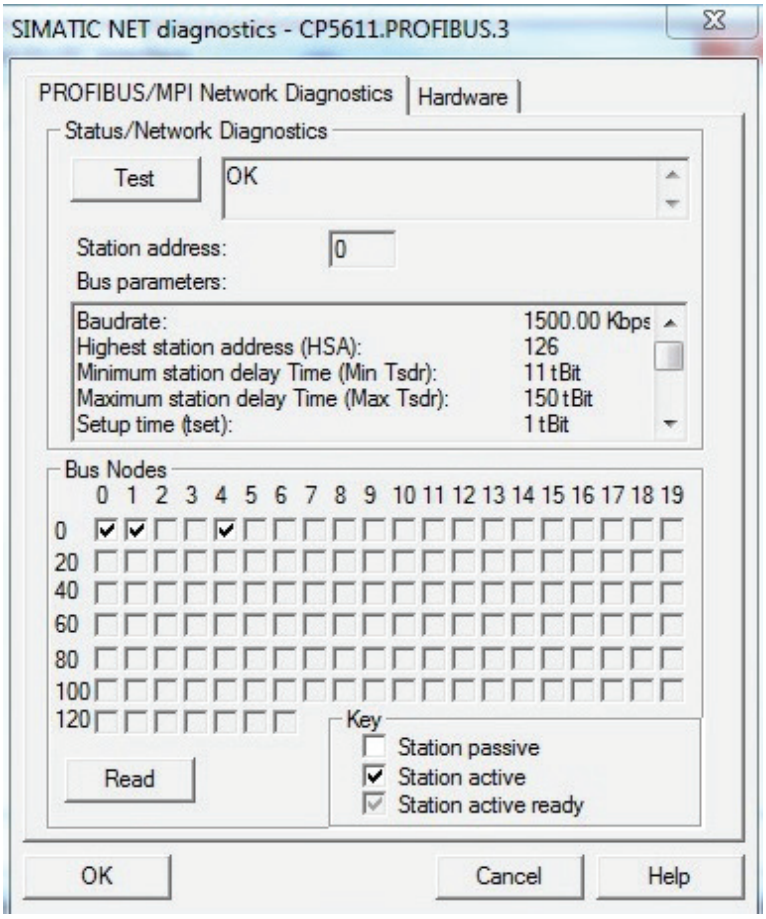


Рис. 7. Конфигурация сегмента сети PROFIBUS DP с тремя устройствами

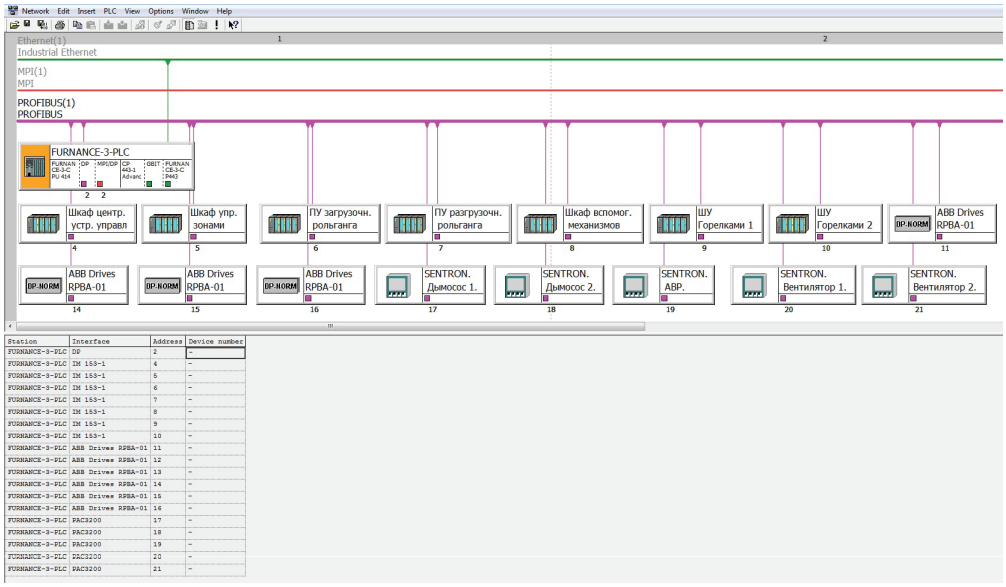
Из рисунков видно, что скорость передачи данных задана 1,5 Мбит/с. Временные параметры заданы в интервалах передачи одного бита информации по шине (tBit). Для данной скорости величина 1tBit будет равна 0,667 мкс. Время, отведенное на опрос каждого устройства (Tslot) составляет 300 tBit. Время инициализации сети (Tset) составляет 1tBit.

Если на шине присутствует только одно ведущее устройство, тогда маркерное кольцо упрощается до конфигурации ведущий-ведомые. Такая конфигурация используется в версии протокола PROFIBUS DP, где ведущее устройство называется DP-master, а ведомые — DP-slaves.

В протоколе DP определены два типа ведущих устройств (класс 1 и класс 2). DP-master класса 1 выполняет функции ведущего устройства по опросу подключенных к нему ведомых устройств, в то время

как DP-master класса 2 обеспечивает дополнительные функции по работе с шиной данных (диагностика шины, параметрирование других устройств на шине и т. п.)

Пример сети PROFIBUS DP с одним ведущим и несколькими ведомыми устройствами изображен на рис. 8. В данной конфигурации контроллер Siemens CPU-414 с коммуникационным модулем CP-443 выступает в роли ведущего устройства с адресом 2. Модули расширения IM153, частотные преобразователи ABB Drives и измерительные приборы SENTRON PAC3200 объединены в один сегмент сети как ведомые устройства с адресами с 4 по 21.



Station	Interface	Address	Device number
FURNANCE-3-PLC	DP	2	–
FURNANCE-3-PLC	IM 153-1	4	–
FURNANCE-3-PLC	IM 153-1	5	–
FURNANCE-3-PLC	IM 153-1	6	–
FURNANCE-3-PLC	IM 153-1	7	–
FURNANCE-3-PLC	IM 153-1	8	–
FURNANCE-3-PLC	IM 153-1	9	–
FURNANCE-3-PLC	IM 153-1	10	–
FURNANCE-3-PLC	ABB Drives RPBA-01	11	–
FURNANCE-3-PLC	ABB Drives RPBA-01	12	–
FURNANCE-3-PLC	ABB Drives RPBA-01	13	–
FURNANCE-3-PLC	ABB Drives RPBA-01	14	–
FURNANCE-3-PLC	ABB Drives RPBA-01	15	–
FURNANCE-3-PLC	ABB Drives RPBA-01	16	–
FURNANCE-3-PLC	PAC3200	17	–
FURNANCE-3-PLC	PAC3200	18	–
FURNANCE-3-PLC	PAC3200	19	–
FURNANCE-3-PLC	PAC3200	20	–
FURNANCE-3-PLC	PAC3200	21	–

Рис. 8. Пример конфигурации сети PROFIBUS в среде разработки программ для ПЛК SIMATIC Manager

4.4. Уровень приложения

Уровень приложения используется только в версии протокола PROFIBUS FMS и состоит из двух частей: FMS (Fieldbus Message Specification) и LLI (Lower Layer Interface).

FMS подуровень описывает коммуникационные объекты, сервисы и модели с точки зрения участников взаимодействия. Цель взаи-

модействия на уровне АСУТП состоит в передаче данных (считывание и запись измеренных значений, загрузка/запуск/останов программ, последовательности событий и т. п.) между двумя участниками взаимодействия. Для взаимодействия между пользовательскими процессами на каждом из двух устройств эти объекты должны быть описаны одинаковым образом. Спецификация FMS описывает перечень поддерживаемых объектов (Object Dictionary). Модель устройства, состоящая из таких объектов, называется Virtual Field Device (VFD). Реализация этой модели в каждом конкретном устройстве выполняется производителем на свое усмотрение и не определяется стандартом PROFIBUS.

К основным функциям подуровня LLI относятся: взаимодействие прикладного (FMS) уровня с канальным (FDL), установление и разрыв соединения, управление соединением, управление потоком данных.

4.5. Версии протокола PROFIBUS

Существует несколько версий протокола PROFIBUS.

PROFIBUS DP (Decentralized Peripherals — децентрализованная периферия). Предназначен для связи промышленного управляющего контроллера с датчиками и исполнительными механизмами. Как правило, использует схему с одним ведущим устройством. Данное устройство контролирует обмен по шине, отправляя запросы и данные для ведомых устройств. Ведомые устройства отвечают только в ответ на запрос ведущего. Данный протокол определен для уровней OSI 1 (RS-485 или Fiber Optic) и 2 (FDL). Уровни с 3 по 7 не используются. В качестве пользовательского интерфейса используется доступ к функциям канального уровня (DP User Interface). Используется для быстрого обмена критичными данными во время автоматизированного управления техническими процессами. Наиболее часто встречающаяся версия протокола в небольших проектах автоматизации.

PROFIBUS PA (Process Automation — управление процессом) — версия протокола, позволяющая подключать к общей сети PROFIBUS датчики и устройства, работающие во взрывоопасных зонах. Для подключения PROFIBUS PA устройств к шине PROFIBUS DP требуется переходник. Данный протокол определен для уровней OSI 1 (IEC 1158–2) и 2 (IEC Interface) (табл. 12). Уровни с 3 по 7 не используются. В качестве пользовательского интерфейса используется

доступ к функциям канального уровня протокола PROFIBUS DP (DP User Interface).

Таблица 12

Характеристики сегмента сети PROFIBUS PA

Параметр	Показатели
Топология сети	«Общая шина», «Звезда», «Дерево»
Среда передачи	Экранированная витая пара
Максимальная длина сегмента	1900 м
Скорость передачи	31.25 кбит/с
Количество узлов	Не более 127 в сети

PROFIBUS FMS (Fieldbus message specification) используется для организации многомастерного режима (multi-master), когда требуется присутствие на шине нескольких ведущих устройств (ПЛК, промышленные компьютеры, системы HMI). Основная особенность — высокоскоростная (до 12 Мбит/с) передача больших объемов данных между интеллектуальными устройствами. Данный протокол определен для уровней OSI 1 (RS-485 или Fiber Optic), 2 (FDL) и 7 (FMS+LLI). Уровни с 3 по 6 не используются (табл. 13).

Таблица 13

Сервисы канального уровня, доступные для каждой версии протокола PROFIBUS

Сервис	Описание	DP	PA	FMS
SDA	Посылка данных с подтверждением			+
SDR	Посылка и запрос данных с ответом	+	+	+
SDN	Посылка данных без подтверждения	+	+	+
CSDN	Циклическая посылка и запрос данных с ответом			+

Контрольные вопросы

1. Какие уровни модели OSI используются в протоколе PROFIBUS?
2. Какие стандарты физического уровня могут быть использованы в протоколе PROFIBUS?
3. Как организован доступ к среде передачи в соответствии с протоколом PROFIBUS?
4. В чем состоит принцип логического маркерного кольца?

5. Какие устройства могут использоваться в качестве ведущих для сети PROFIBUS-устройств?
6. В чем отличие версий протокола PROFIBUS DP, PROFIBUS PA и PROFIBUS FMS друг от друга?
7. Для чего предназначена и какие особенности имеет версия PROFIBUS FMS?

5. Протоколы АСУТП на базе стандарта ETHERNET

5.1. Обзор технологии Ethernet с точки зрения промышленных сетей

В настоящее время подавляющее большинство сетей для передачи данных общего назначения построены на основе стандартов группы Ethernet (IEEE 802.3). Однако в области промышленной связи использование этого стандарта ограничено его фундаментальными недостатками.

В технологическом процессе ключевым требованием является привязка получаемой и передаваемой по сети информации к этапам работы оборудования. Это, в свою очередь, требует от используемого протокола введения гарантированного времени доставки данных, укладывающегося в допустимый для управления процессом диапазон быстродействия. При этом заданное время отклика должно обеспечиваться для всех режимов нагрузки сетевой инфраструктуры и оконечных устройств, обменивающихся информацией.

По аналогии с операционными системами можно выделить сети жесткого и мягкого реального времени:

- сеть жесткого реального времени обеспечивает передачу данных за гарантированное время для любых внешних условий;
- сеть мягкого реального времени обеспечивает передачу данных за гарантированное время в среднем за время работы сети.

К примерам системы жесткого реального времени можно отнести протокол PROFIBUS, в котором имеется гарантированное время оборота маркера, ограничивающее время отклика любого устройства в системе, а также гарантирующее каждому устройству возможность получения и передачи данных в заданный интервал времени.

Поддержка реального времени должна выполняться не только сетевой инфраструктурой, но и программным обеспечением устройств, т. к. именно от времени реакции каждого устройства в сети зависит общее время отклика системы на поступающие от оборудования данные о технологическом процессе. С этой целью в контроллерах используются операционные системы реального времени.

Стандарт Ethernet привлекателен в первую очередь своей скоростью: даже относительно старая версия Fast Ethernet имеет скорость передачи 100 Мбит/с, в то время как самые быстрые версии протокола PROFIBUS ограничивают скорость передачи 12 Мбит/с.

Другим преимуществом является хорошо проработанный и широко используемый стек протоколов: протоколы семейства TCP/IP предоставляют универсальную базу для вышележащих протоколов, ориентированных на решение пользовательских задач.

Фундаментальным недостатком Ethernet является отсутствие гарантированного времени передачи между устройствами. Это обусловлено методом доступа к общей среде передачи, описанным в стандарте 802.3: CSMA/CD — множественный доступ с контролем несущей и обнаружением коллизий. В соответствии с описанием доступ к среде устройства получают на конкурентной основе, забирая освободившуюся после предыдущего обмена среду в монопольное распоряжение. Если в данный момент устройство обнаруживает, что среда занята (обнаружена несущая), то устройству остается ждать неопределенно долгое время до тех пор, пока передающее устройство не освободит среду. Стандартом не накладываются ограничительные требования на промежуток владения средой передачи. Другим аспектом является то, что если несколько устройств начали передачу одновременно, в среде возникает коллизия, и передаваемые данные оказываются испорченными. В этом случае стандарт предписывает устройствам прервать передачу и выждать случайный интервал времени, после чего повторить попытку передачи. Все эти особенности делают невозможным оценку гарантированного времени доставки сообщения между узлами такой сети.

Вследствие этого в последнее время предпринимаются попытки модифицировать оборудование и программную часть Ethernet-совместимых устройств для использования их как базы промышленных сетей. Общее название таких модификаций — Real-Time Ethernet.

Основные направления по достижению реального времени можно сформулировать следующим образом:

- использование полного стека Ethernet без модификации. В этом случае неопределенность и отсутствие гарантированного времени доставки компенсируются скоростью передачи;
- решения на базе стека протоколов TCP/IP. Данные решения оставляют неизменным коммуникационный стек Ethernet протоколов нижних уровней (с физического по транспортный). Вышележащие уровни модифицируются таким образом, чтобы эмулировать real-time обмен, как правило, с помощью синхронизации времени передатчика и приемника. Возможности такого подхода существенно ограничены, вследствие того, что неопределенность заложена на канальном уровне в методе CSMA/CD;
- модификация всех уровней модели OSI для поддержки протокола реального времени. В этом случае можно достичь соответствия всем требованиям вплоть до жесткого реального времени. Недостатком является потеря совместимости с большинством устройств, поддерживающих стандарт Ethernet (использование специализированных коммутаторов, сетевых карт и т. п.).

Механизм доступа к среде также претерпевает существенные изменения для достижения гарантированного времени отклика:

- централизованное управление доступом. Как правило, реализуется в виде схемы «ведущий-ведомые» и заключается в монополизации доступа к среде передачи одним устройством;
- централизованное управление путем синхронизации часов всех устройств в сети от одного сервера (рассылка меток времени);
- децентрализованное управление с помощью системы специализированных сетевых устройств, работающих на канальном уровне.

5.2. Стандарт PROFINET

Стандарт Profibus Network — это версия протокола PROFIBUS, расширяющая его использование на сети, совместимые со стандартом Ethernet [13]. Разработка и расширение данного стандарта выполняются консорциумом фирм Profibus&Profinet International), деятельное участие в котором принимают специалисты компании Siemens.

Часть протокола, регламентирующая доступ к удаленной периферии простых устройств, получила название PROFINET I/O (Input/Output). В рамках этой концепции все устройства в сети разделены на управляющие контроллеры (IO-Controller), подчиненные устройства (IO-Devices) и устройства для обслуживания сети (IO-Supervisors). Протокол описывает взаимодействие всех трех классов устройств между собой.

Вторая часть протокола носит название PROFINET CBA (Component Based Automation), описывает взаимодействие между собой модульных систем управления и предназначена для объединения между собой интеллектуальных устройств и целых систем.

Существует три режима работы протокола в зависимости от имеющегося оборудования и требований:

- PROFINET TCP/IP — для приложений, не требующих передачи в реальном времени. Примерное время передачи данных в этом режиме составляет около 100 мс. Может использоваться для связи с HMI-устройствами, а также для сбора статистики, администрирования и настройки сети.
- PROFINET RT (Real-Time) — протокол для обеспечения мягкого реального времени, или без требований к времени доставки данных. Обеспечивает время цикла до 1 мс. Данный режим использует фреймы Ethernet с дополнительным VLAN-заголовком в соответствии со спецификацией 802.1Q. Благодаря этому заголовку, а именно флагу приоретизации трафика, входящему в него, кадры PROFINET получают высокий приоритет при передаче по сети, по сравнению со сторонним трафиком. Так как данные заголовки реализованы на канальном уровне модели OSI, передача кадров PROFINET RT через межсетевые устройства (роутеры) невозможна.
- PROFINET IRT (Isochronous Real-Time) — протокол для обеспечения жесткого реального времени. Обеспечивает время цикла менее 1 мс. Для его использования требуется аппаратная поддержка на каждом из подключенных к сети узлов. Обмен данными происходит жестко тактируемыми циклами. Точность синхронизации циклов на каждом устройстве в сети поддерживается с помощью аппаратного контроллера сетевого интерфейса, а также контроллеров промежуточных узлов, формирующих топологию сети (коммутаторов). Внутри каждого цикла обмена существу-

ет изохронная часть (время передачи данных в режиме жесткого реального времени) и неизохронная часть (время передачи данных в режиме мягкого реального времени с использованием стека TCP/IP) (рис. 9).

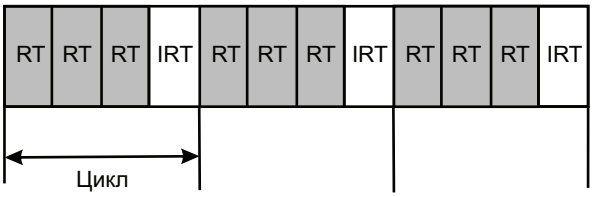


Рис. 9. Распределение RT- и IRT-траффика в сети PROFINET

Существует два режима IRT обмена: IRT (High Flexibility) — с высокой гибкостью и IRT (High Performance) — с высокой производительностью. IRT (High Flexibility) использует выделенный слот времени для передачи изохронных (требовательных ко времени) доставки данных. IRT (High Performance) в дополнение к выделенному слоту времени использует для увеличения точности строго заданный порядок обмена устройств, учитывающий существующую топологию сети. Знание топологии позволяет определить требуемый резерв пропускной способности для каждой линии (каждого участка кабеля). Вследствие этого величина IRT-слота времени может быть уменьшена, что позволяет увеличить время для остальных участников обмена и повысить производительность работы сети по сравнению с режимом IRT (High Flexibility).

Сравнение версий протокола PROFINET представлено в табл. 14.

В режиме PROFINET IRT синхронизация времени узлов выполняется с использованием протокола IEEE 1588 — Стандарт протокола синхронизации точного времени для сетевых измерительных систем и систем управления (бывший стандарт PTP). Этот протокол позволяет достичь точности синхронизации устройств менее микросекунды в локальных сетях. Задача протокола — синхронизировать часы устройства (ведомого) с какими-либо эталонными часами (например, часами устройства, принятого за эталон). Такое устройство называется в протоколе ведущим и может иметь независимую систему поддержания точности (например, по спутнику). Фактически задача состоит в вычислении смещения Δt часов ведомого устройства относительно

ведущего. Эта задача осложняется тем, что оба устройства могут обмениваться данными только по сети, которая вносит свою задержку в виде времени доставки сообщения $t_Д$.

Таблица 14

Сравнение версий протокола PROFINET IO

Параметр	PROFINET RT	PROFINET IRT (High Flexibility)	PROFINET IRT (High Performance)
Real time class	Класс 1	Класс 2	Класс 3
Режим передачи	Приоритетность RT-кадров за счет использования VLAN-заголовка	Резервирование полосы пропускания, т. е. выделение слота времени только для IRT-коммуникаций	Резервирование полосы пропускания с использованием жесткого порядка обмена, основанного на известной топологии системы
Точность времени обмена	Время обмена варьируется из-за использования TCP/IP-пакетов	Гарантированная точность за счет резервирования слота времени для IRT данных	Время обмена можно определить с высокой точностью исходя из известной топологии сети
Изохронные приложения	Не поддерживается	Не поддерживается	Поддерживается
Аппаратная поддержка, использующая специализированные коммутаторы Ethernet	Нет	Да	Да

Для аппаратной поддержки стандарта требуется, чтобы устройства могли зафиксировать и сохранить время прихода кадра данных из сети на приемный интерфейс. Таким образом, момент приема отмечается временной меткой, которая в дальнейшем будет использована для вычисления смещения (рис. 10). Данную метку можно получить и без аппаратной поддержки, однако в этом случае точность ее вычисления существенно снижается. Это происходит из-за неконтролируемого промежутка времени между приходом кадра на интерфейс устройства и моментом вызова обработчика прерывания по этому событию, в котором будет зафиксирована нужная метка времени прибытия кадра.

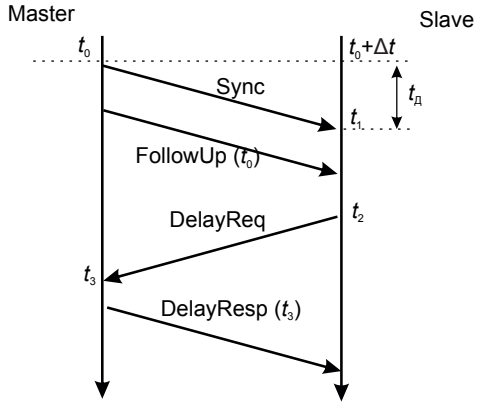


Рис. 10. Диаграмма обмена при взаимодействии по протоколу IEEE 1588

Для определения смещения ведущее устройство периодически отправляет широковещательные сообщения Sync, записав при этом временную метку t_0, которая показывает момент отправки данного сообщения. Ведомое устройство получает данное сообщение и фиксирует момент его прихода меткой t_1. Вслед за сообщением Sync ведущее устройство отправляет сообщение FollowUp, в которое вкладывает временную метку t_0. Получив данное сообщение, ведомое устройство может вычислить сумму $\Delta t + t_Д$ из выражения

$$t_1 = t_0 + \Delta t + t_Д \Rightarrow \Delta t + t_Д = t_1 - t_0. \quad (1)$$

Затем необходимо вычислить величину $t_Д$, чтобы исключить ее из полученной суммы.

Для этого ведомое устройство отправляет ведущему сообщение DelayReq, сохраняя у себя момент его отправки t_2. Ведущее устройство фиксирует у себя момент прихода этого сообщения t_3 и отправляет его внутри ответного сообщения DelayResp. Величины t_2 и t_3 связаны соотношением

$$t_3 - (t_2 - \Delta t) = t_Д. \quad (2)$$

Из формул (1) и (2) можно выразить искомое смещение времени

$$\Delta t = [(t_1 - t_0) - (t_3 - t_2)]/2. \quad (3)$$

Зная смещение Δt, ведомое устройство может подстроить свои часы для устранения рассинхронизации с часами ведущего.

Пример конфигурации сети PROFINET с ведущим контроллером SIMATIC S7–1500 и несколькими ведомыми устройствами (интерфейсные модули IM155 и панель оператора KTP700) представлен на рис. 11. Среди представленных ведомых устройств панель оператора не поддерживает ни режим RT, ни IRT, интерфейсные модули и ПЛК могут работать как в режиме RT, так и в IRT. По классификации PROFINET IO ПЛК относится к группе IO Controller, интерфейсные модули — к группе IO Devices.

Настройка сети начинается с построения топологии на физическом уровне (в окне Network view — рис. 11).

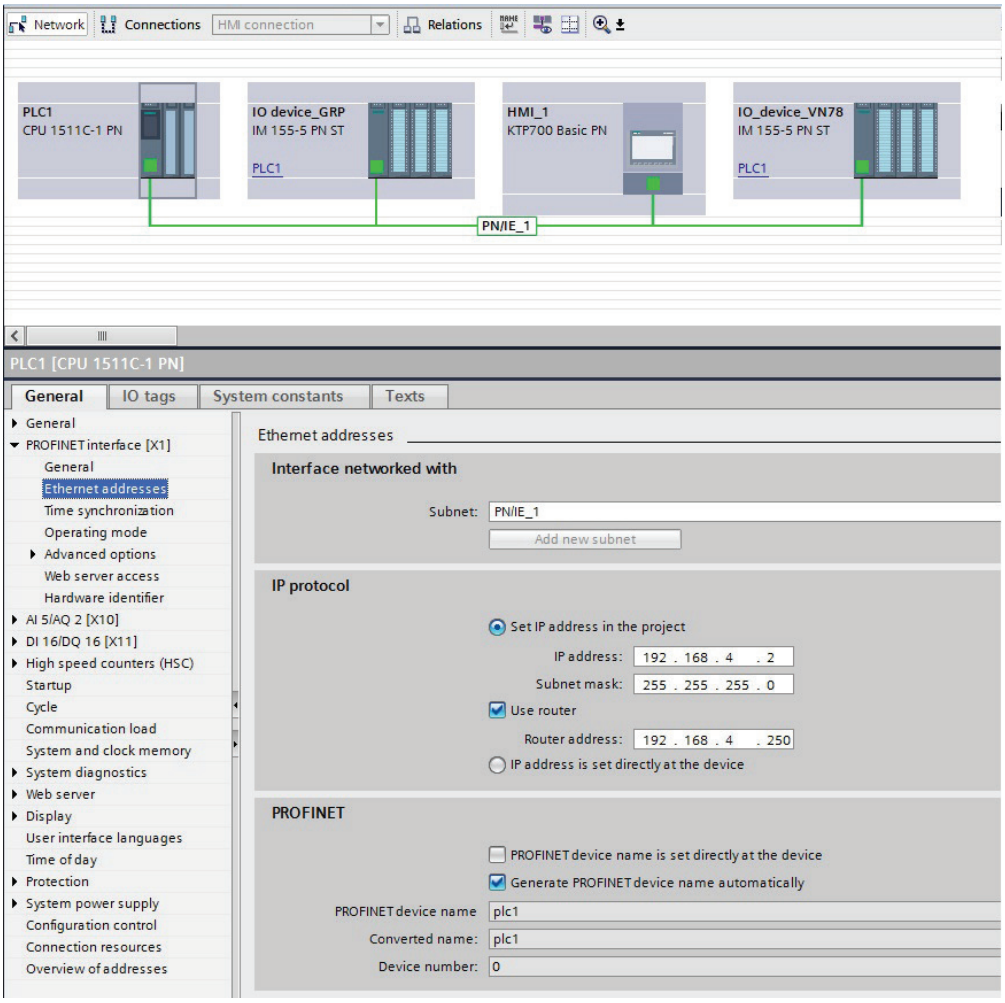


Рис. 11. Пример конфигурации сети PROFINET

На данном экране необходимо показать, что все устройства используют одну сетевую инфраструктуру (объединить их с помощью сегмента сети PROFINET).

Предположим, что нам требуется, чтобы интерфейсный модуль IO device_GRP функционировал в режиме IRT, а другой (IO device_VN78) — в режиме RT. Для устройств, объединяющихся в сегмент IRT необходимо явно указать топологию их соединений. Это делается на вкладке Topology view (рис. 12). Здесь необходимо указать, какие устройства соединены между собой и какими портами.

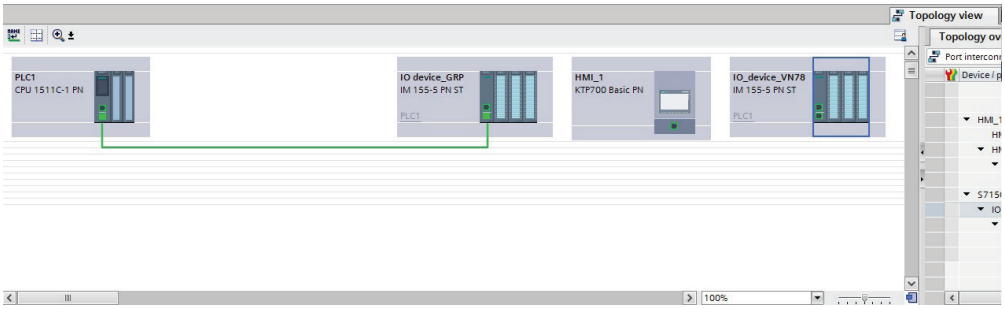


Рис. 12. Явное указание топологии для объединения устройств в режиме IRT (High Performance)

После этого необходимо присвоить адреса ведущему устройству. На рис. 11 видно, что ПЛК PLC1 присвоены следующие адреса:

IP адрес — 192.168.4.2,

PROFINET device name — plc1,

Device Number — 0 (присваивается нулю для устройств типа IO Controller).

В случае, когда сеть PROFINET соединена с внешней Ethernet-сетью маршрутизатором, необходимо указать адрес маршрутизатора. Это может потребоваться, например, если из внешней сети SCADA-система обращается за данными к ПЛК по NRT-протоколу.

Настройки RT-режима PROFINET задаются в группе Advanced options — Real Time settings (рис. 13) Наиболее важное значение имеет поле Send clock. Это время между двумя последовательными циклами передачи IRT- или RT-данных. Другими словами, это минимальное время цикла обмена ведущего устройства с ведомым (см. интервал Цикл на рис. 9). В данном случае оно указано равным 1 мс. Для изменения данного значения необходимо нажать кнопку Domain settings,

выводящую настройки всего сегмента сети PROFINET. При желании можно изменять данное время в пределах от 0,250 до 4 мс. Видно, что ПЛК будет ведущим устройством для синхронизации времени (Sync-master) по протоколу IEEE 1588. А также, что он поддерживает работу в режимах как RT, так и IRT. На основании количества подключенных ведомых устройств было вычислено оптимальное время слота для IRT-данных (Calculated bandwidth for cyclic IO data): 0,011 мс.

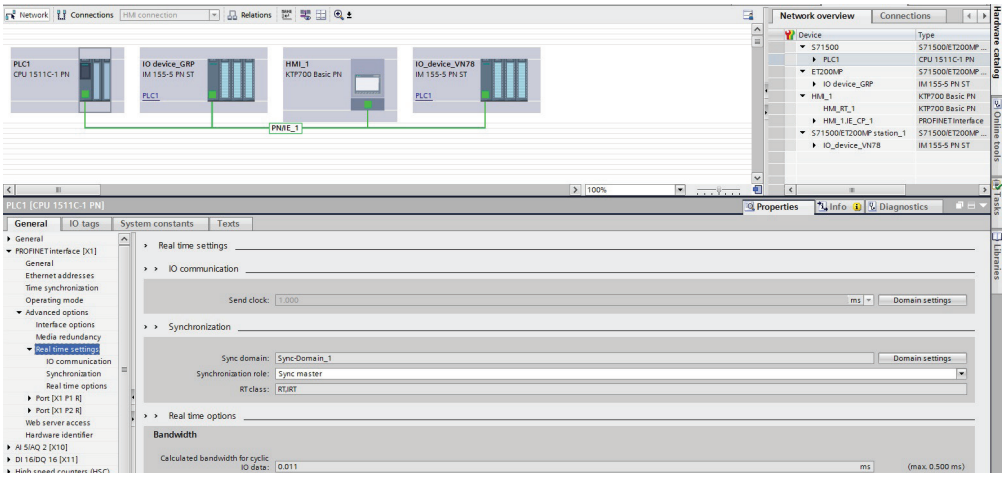


Рис. 13. Настройки Real-Time режима на ведущем устройстве

После настройки ведущего устройства необходимо выполнить настройку ведомых устройств (рис. 14) Для примера рассмотрим настройку подключения к сети PROFINET интерфейсного модуля IM155. Для включения устройства в сеть необходимо задать ему IP-адрес, а также PROFINET device name и Device Number (выбираются произвольным образом с учетом требований уникальности внутри одного сегмента PROFINET).

Кроме общего для всех устройств сегмента времени цикла (send clock), для каждого устройства в режиме IO Device нужно указать время опроса (Update time). Это время кратно времени цикла и показывает, как часто данное устройство отправляет данные контроллеру. В этом случае видно, что время опроса установлено равным 2 мс, что означает передачу данных от устройства к контроллеру каждый второй цикл обмена данными. В группе Synchronization указывается, в каком из режимов будет работать устройство: RT или IRT, а следовательно, и в каком слоте времени оно будет передавать свои данные.

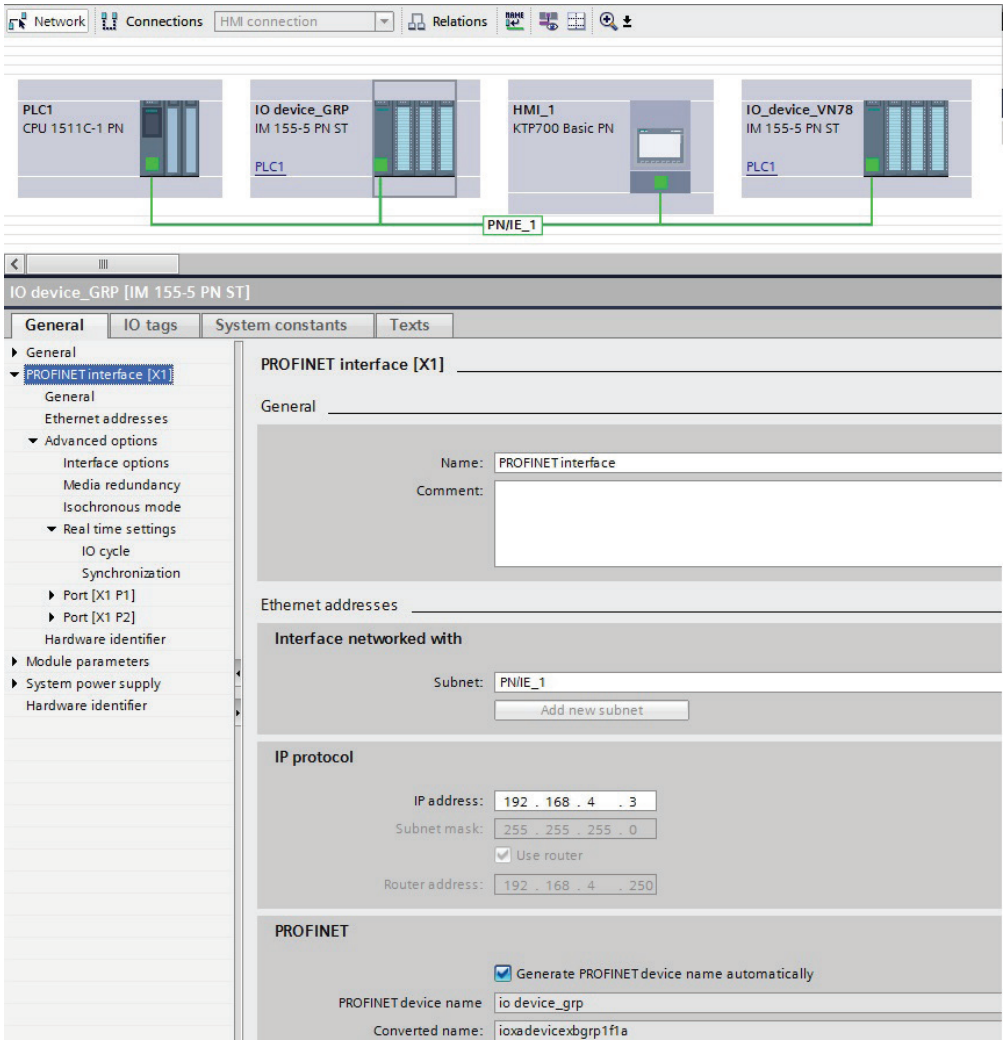


Рис. 14. Настройка ведомого устройства в сети PROFINET

Выбор режима работы задается в группе Advanced options — Real-Time settings (рис. 15).

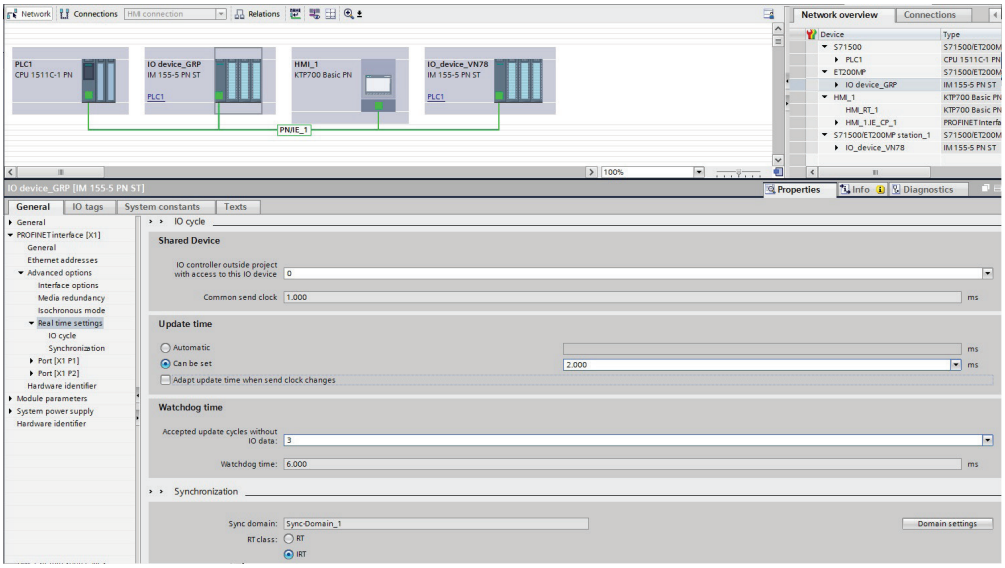


Рис. 15. Настройка работы в режиме IRT в сети PROFINET

В случае выбора работы устройства в изохронном режиме в группе свойств Isochronous mode (рис. 16) можно посмотреть вычисленные времена:

- Time Ti (в примере 0,01814 мс) — время, необходимое устройству для приема пакета данных от контроллера (зависит от объема данных в устройстве). Для интерфейсного модуля определяется числом дискретных и аналоговых выходов на нем;
- Time To (в примере 0,017508 мс) — время, необходимое устройству для передачи пакета данных контроллеру (зависит от объема данных в устройстве). Для интерфейсного модуля определяется числом дискретных и аналоговых входов на нем.

Также внизу изохронный режим можно включить для каждого из имеющихся слотов интерфейсного модуля.

Совокупную информацию о работе сегмента сети PROFINET можно увидеть в окне его свойств (рис. 17).

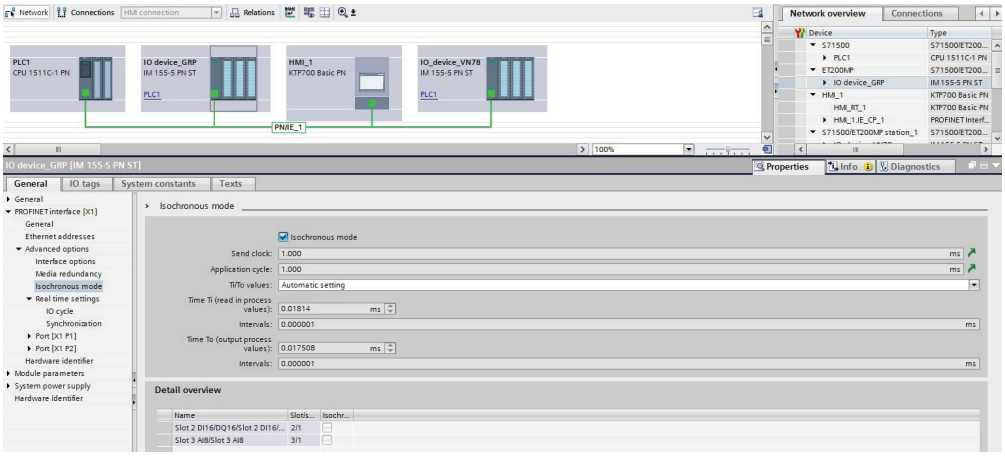


Рис. 16. Настройка изохронного режима обмена для устройства типа IO Device PROFINET

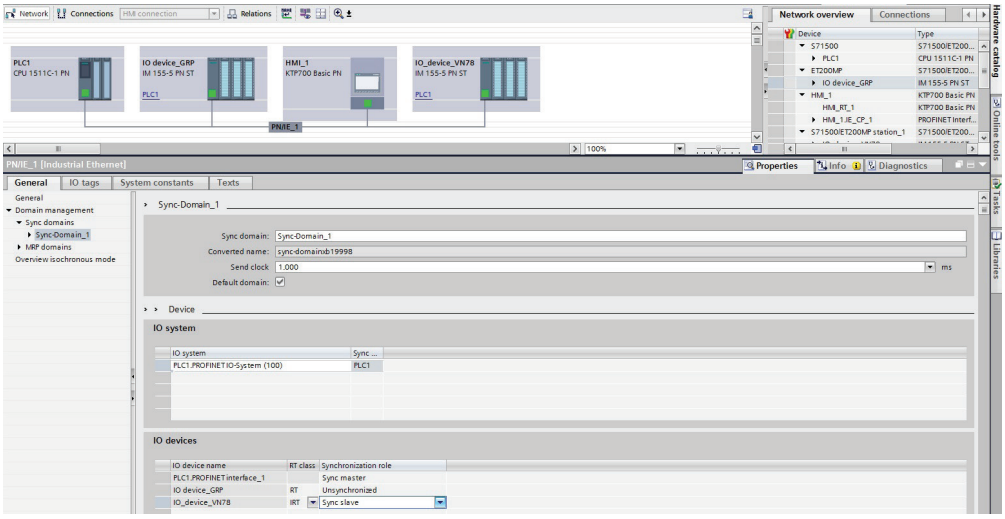


Рис. 17. Общая информация о работе сегмента сети PROFINET

На данной странице можно задать время цикла для сегмента сети, выбрать ведущее устройство для протокола синхронизации времени IEEE 1588, задать режим работы (RT или IRT) для ведомых устройств, а также назначить или отключить синхронизацию времени на них.

В группе Sync domains можно увидеть долю используемой полосы пропускания линии (в мс), которая занята RT-данными и IRT-данными (рис. 18).

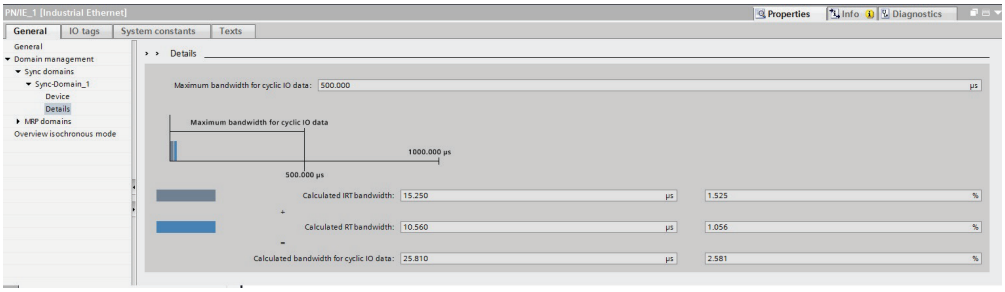


Рис. 18. Использование пропускной способности линии

Видно, что при времени цикла, равном 1 мс (1000 мкс), на передачу данных в режиме RT отводится 15,250 мкс (или 1,525 % от пропускной способности), в то время как на передачу данных от устройств IRT отводится 10,560 мкс (что составляет 1,056 % пропускной способности линии). При этом на циклическую передачу данных в совокупности отведено 500 мкс за цикл (50 % пропускной способности).

5.3. Протокол POWERLINK

Данный протокол для передачи данных в реальном времени по сетям Ethernet был разработан австрийской компанией B&R в 2001 г. (с 2017 г. — подразделение ABB). В настоящее время стандарт открыт и поддерживается группой Ethernet Powerlink Standardization Group (EPSG).

В качестве физической среды передачи стандарт использует витую пару (стандарты 100Base-TX и 1000Base-T). Для соблюдения требований реального времени необходимо, чтобы в сегменте сети не было посторонних устройств (не включенных в сеть POWERLINK), вносящих неконтролируемые задержки при передаче.

На канальном уровне доступ к шине регламентируется с помощью организации циклов обмена фиксированной длительности. За синхронизацию и организацию циклов отвечает ведущее устройство (Managing Node). Внутри каждого цикла предусмотрен слот времени для передачи данных в режиме реального времени (изохронная передача), а также слот данных для асинхронной передачи данных. Начало каждой фазы объявляется ведущим устройством в виде широковеща-

тельного сообщения ведомым устройствам (Controlled Nodes). В начале изохронной части ведущее устройство генерирует кадр SoC (Start of Cycle) для синхронизации ведомых устройств.

После рассылки кадра SoC для обмена с конкретным ведомым устройством ведущее генерирует кадр начала обмена, который называется Poll Request (запрос). Выбранное ведомое устройство отвечает на запрос кадром Poll Response (ответ). Таким образом происходит последовательный по времени опрос всех ведомых устройств в сети. Асинхронная часть цикла начинается с кадра SoA (Start of Asynchronous), выдаваемого широковещательно ведущим устройством, после чего выполняется обмен некритичными, с точки зрения гарантии времени доставки, данными. В фазе асинхронного обмена выполняется обмен пакетами, совместимыми со стеком TCP/IP, что позволяет использовать в сети POWEWRLINK обычное сетевое оборудование (коммутаторы и маршрутизаторы). Однако внутри сегмента, поддерживающего жесткое реальное время, не рекомендуется использовать никакое сетевое оборудование, за исключением концентраторов, с целью уменьшения задержек передачи. Также предполагается, что сегмент реального времени должен быть отделен от остальной части сети маршрутизатором.

В последних версиях стандарта POWERLINK появилась поддержка синхронизации нескольких сегментов реального времени друг с другом посредством протокола IEEE 1588 (см. п. 5.2).

В протоколе POWERLINK используются широковещательные посылки в сегменте Ethernet (рис. 19): каждый узел может выдавать свои данные в сеть с помощью широковещательных сообщений, и любой другой узел, как правило, может принимать эти посылки. Принимающие узлы используют адрес назначения, включенный в пакет данных, чтобы определить, предназначен ли данный пакет для них или нет. Этот принцип передачи обеспечивает прямое взаимодействие между участниками (без маршрутизирующих устройств). Эта техника может использоваться, например, для синхронизации вращения нескольких приводов.

Ведомые устройства POWERLINK подразделяются на два класса в зависимости от требований к ответам на запросы ведущего:

- Постоянные. Ведомые устройства этого класса обмениваются данными с ведущим устройством в каждом цикле обмена по сети.
- Мультиплексированные. Данные устройства могут передавать свои данные не в каждом цикле обмена.

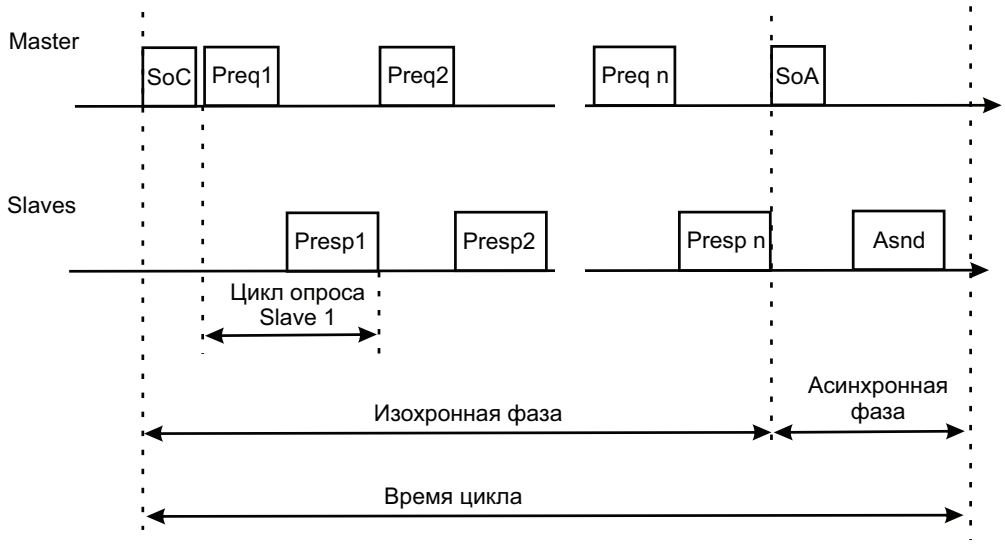


Рис. 19. Диаграмма обмена при взаимодействии по стандарту POWERLINK

Опрос мультиплексированных узлов разбит на несколько циклов обмена. Это позволяет опрашивать в изохронной фазе обмена большее количество устройств за счет временного исключения некоторых устройств из опроса, но уменьшает частоту опроса таких устройств.

На рис. 20 показана диаграмма опроса ведомых устройств S1-S8 ведущим устройством.

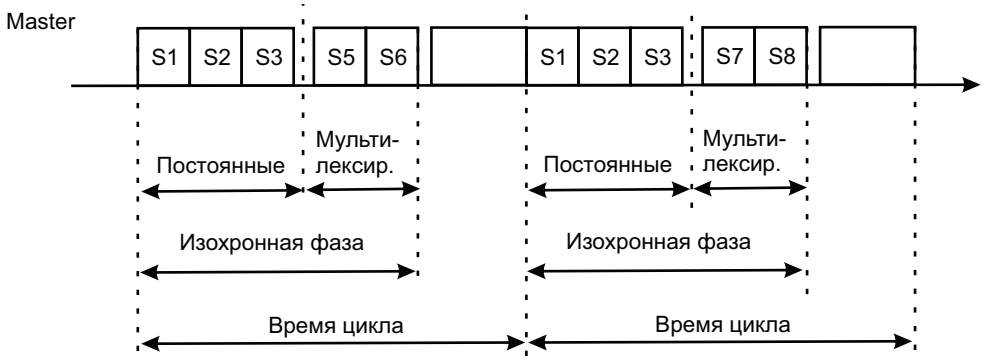


Рис. 20. Пример мультиплексирования при опросе ведомых устройств POWERLINK

Видно, что устройства S1, S2, S3 опрашиваются в каждом цикле обмена, в то время как устройства S5, S6, S7, S8 опрашиваются в од-

ном цикле из двух. Это уменьшает скорость обновления информации от устройств S5-S8, однако позволяет не увеличивать общее время цикла.

Каждый узел POWERLINK (MN, CN или маршрутизатор) должен иметь уникальный идентификатор узла в сегменте POWERLINK.

Номер 240 постоянно закреплен за MN, поэтому узел, которому присвоен этот адрес, должен выполнять функции ведущего устройства.

Адреса с 1 по 239 могут использоваться для ведомых устройств. Адрес может быть задан как при конфигурировании устройства, так и установлен аппаратно (например, в виде набора переключателей). Термины одноадресная, многоадресная и широковещательная передача относятся к адресам POWERLINK. Если используется широковещательный адрес POWERLINK, то содержащий его кадр Ethernet должен иметь признак широковещательной рассылки (MAC-адрес назначения равный FF-FF-FF-FF-FF-FF).

Адрес 0 не разрешено использовать для адресации узлов.

Адреса с 241–250 зарезервированы.

Адрес 251 предназначен для адресации узла к самому себе (петлевой адрес). Этот адрес не должен назначаться реальным узлам.

Адрес 252 не используется (dummy).

Адрес 253 присваивается диагностическому устройству. Это устройство должно быть доступно остальным узлам без какой-либо конфигурации. Доступ к нему может быть только асинхронным.

Устройство с адресом 254 служит шлюзом между POWERLINK сегментом сети и Ethernet сетью общего назначения.

Адрес 255 широковещательный.

Формат кадра изображен в табл. 15, 16.

Таблица 15

Формат кадра POWERLINK Канального уровня

Смещение от начала кадра, б	Назначение	Описание	Принадлежность поля
0...5	MAC-адрес узла назначения	–	Ethernet II
6...11	MAC-адрес узла отправителя	–	Ethernet II
12...13	EtherType	Должен содержать значение 88ABh	Ethernet II
14	MessageType	Тип сообщения (табл. 16)	POWERLINK

Окончание табл. 15

Смещение от начала кадра, б	Назначение	Описание	Принадлежность поля
15	Destination	POWERLINK адрес узла назначения	POWERLINK
16	Source	POWERLINK адрес узла отправителя	POWERLINK
17...*n*	Data	Данные зависят от значения поля MessageType	POWERLINK
n+1... *n*+4	CRC	Контрольная сумма кадра	Ethernet II

Таблица 16

Значение поля MessageType

MessageType	Обозначение	MAC-режим отправки
Start of Cycle	SoC	Multicast
PollRequest	PReq	Unicast
PollResponce	PRes	Multicast
Start of Asynchronous	SoA	Multicast
Asynchronous Send	ASnd	Multicast

Организацией разработчиком протокола выпущена реализация программного обеспечения как ведущего, так и ведомых устройств на языке ANSI C — openPOWERLINK, что позволяет скомпилировать его под любую из существующих аппаратных платформ. Данный продукт выпущен под лицензией BSD, которая предоставляет разрешение на бесплатное использование, распространение, изменение и улучшение программного обеспечения. Разработчик прикладного ПО также может интегрировать этот код в свой коммерческий продукт.

5.4. Протокол EtherNet/IP

Данный протокол был разработан в 2000 г. компанией Allan-Bradley и в данный момент поддерживается и развивается ассоциацией компаний Open DeviceNet Vendors Association, Inc. (ODVA). Данная ассоциация, начиная с 1995 г. разрабатывает свой протокол для поддержки промышленных коммуникаций — CIP (Common Industrial Protocol). Данный протокол является транспортно-независимым и описывает

взаимодействие между собой промышленных устройств, используя верхние уровни модели OSI (начиная с сеансового). Взаимодействие устройств описывается протоколом в виде объектной модели. Высокоуровневость протокола позволяет использовать его поверх имеющейся транспортной инфраструктуры сетей передачи данных, в том числе и поверх сетей Ethernet. Протокол EtherNet/IP является адаптацией протокола CIP для использования с этой группой сетевых стандартов. В качестве базы данный протокол использует стеки протоколов TCP и UDP, в пакеты которых вкладываются кадры управления и данных протокола CIP. Для данных, не требующих гарантий времени доставки, используется передача с помощью TCP-пакетов. Для данных, требующих доставки в реальном времени, используется протокол датаграмм UDP. В первом случае достигается повышенная надежность при передаче, а во втором увеличивается скорость доставки за счет отсутствия посылок подтверждения узлу-отправителю пакета.

Протокол поддерживает различные режимы обмена данными: запрос-ответ, циклический опрос, уведомление о наступлении события.

Для синхронизации времени узлов используется подмножество протокола CIP — CIPSync, реализующее алгоритм стандарта синхронизации IEEE 1588 (см. п. 5.2).

Стандарт описывает два типа взаимодействия: обмен явными (explicit) и неявными (implicit) сообщениями (табл. 17).

Таблица 17

Типы взаимодействия узлов по протоколу Ethernet/IP

Тип сообщений	Взаимодействие по сети	Протокол транспортного уровня	Форма взаимодействия	Типичная область использования	Примеры
Явное	С установлением соединения или без него	TCP/IP	Запрос/ответ	Передача данных, не требовательных к времени доставки	Чтение/запись параметров конфигурации
Неявное	С установлением соединения	UDP/IP	Передача данных	Передача данных в режиме реального времени	Данные для управления удаленным объектом

Обмен явными сообщениями характерен для ситуаций типа «запрос/ответ» (например, для общения клиента с сервером). Этот тип сообщений используется для данных, передача которых не требует

гарантированного времени доставки (реального времени). Явные сообщения включают в себя описание их назначения, поэтому передавать информацию с их помощью можно медленнее, но формат такой информации поддерживает больше возможностей. Например, они могут использоваться для систем человеко-машинного интерфейса и систем сбора данных, а также для удаленного конфигурирования устройств. В терминологии CIP-протокола при явном обмене сообщениями клиент отправляет запрос к службе удаленного объекта (сервера), например для чтения или записи данных. В данном типе сообщений используется протокол TCP. Явные сообщения в протоколе CIP могут использоваться как с предварительным установлением CIP-соединения, так и без него.

Неявный обмен сообщениями (называемыми часто сообещениями «ввод-вывод» (I/O)) критичен ко времени доставки информации. Этот тип взаимодействия используется для обмена данными в реальном времени, где скорость и низкая задержка передачи играют важную роль. Неявные сообщения содержат очень мало служебной информации, поэтому передача с их помощью более эффективна, но менее конфигурируема, чем с помощью явных сообщений. Для обмена неявными сообщениями требуется предварительно установить соединение (CIP-соединение). При этом согласовывается и скорость передачи пакетов, а также формат данных. EtherNet/IP использует транспортный протокол UDP для неявного обмена сообщениями. При этом могут быть выбраны как одноадресная, так и многоадресная рассылки UDP-пакетов.

В протоколе описаны следующие типы устройств:

- сервер явных сообщений (explicit message server): данное устройство отвечает на запросы, отправленные клиентом в режиме “запрос/ответ”. Примером такого устройства является считыватель штрих-кода;
- клиент явных сообщений (explicit message client): программа, отправляющая запрос к серверу для получения данных. Частота обращений и требования к задержкам, как правило, не критичны. Примерами таких программ являются устройства HMI, средства программирования, которые собирают данные с устройств управления;
- адаптер ввода-вывода (I/O Adapter): устройство, которое располагает данными и выводит их в режиме реального времени для сканера ввода/вывода. Адаптер ввода-вывода, как правило, также является сервером явных сообщений; может представлять со-

бой как простое устройство удаленной периферии (ввода/вывода сигналов), так и интеллектуальное управляющее устройство. Например — система пневматического клапана;
- сканер ввода-вывода (I/O Scanner): данное устройство устанавливает соединение с адаптерами ввода-вывода и получает от них информацию в виде неявных сообщений. Обычно в качестве данного устройства выступает программатор или программируемый контроллер. Кроме задачи получения данных, в задачу сканера входят конфигурирование и настройка остальных устройств сети EtherNet/IP, поэтому он может выступать и в качестве сервера явных сообщений.

Вцелом использование нижних уровней стандарта Ethernet без изменений делает протокол EtherNet/IP пригодным лишь для управления системами в режиме мягкого реального времени. Неконтролируемые задержки в доставке пакетов UDP не могут быть полностью скомпенсированы процедурами вышележащих уровней. Можно лишь минимизировать их, придерживаясь некоторых рекомендаций:
- устранить весь посторонний траффик в сегменте сети;
- минимизировать широковещательные посылки путем настройки коммутаторов сети (отключение алгоритмов Spanning Tree, ARP и подобных);
- повышение приоритетности траффика между узлами путем выделения его в отдельный VLAN или маркировки флагами повышенного приоритета.

5.5. Протокол EtherCAT

Протокол EtherCAT [14] (Ethernet for Control Automation Technology) был разработан компанией Bechoff Automation и может использоваться для достижения как жесткого, так и мягкого реального времени доставки сообщений.

В отличие от стандартного Ethernet, протокол EtherCAT по-другому использует имеющиеся физические линии связи. Если в стандартном сегменте линии, подсоединенные к каждому устройству сети, используются только для связи с этим устройством посредством коммутатора, то в протоколе EtherCAT все физические линии в сегменте обра-

зуют логическое кольцо, информация по которому всегда передается только по заранее заданному маршруту.

В логическом кольце циркулирует один кадр, совместимый с кадром канального уровня Ethernet (поле Ethertype = 0×88a4). Внутри кадра каждому устройству в сети заранее отведен определенный участок поля данных, в который устройство записывает свои данные или считывает предназначенные ему. Данный кадр транслируется сквозным образом каждым устройством сети (с входного интерфейса в выходной) и запись/чтение выполняются на лету.

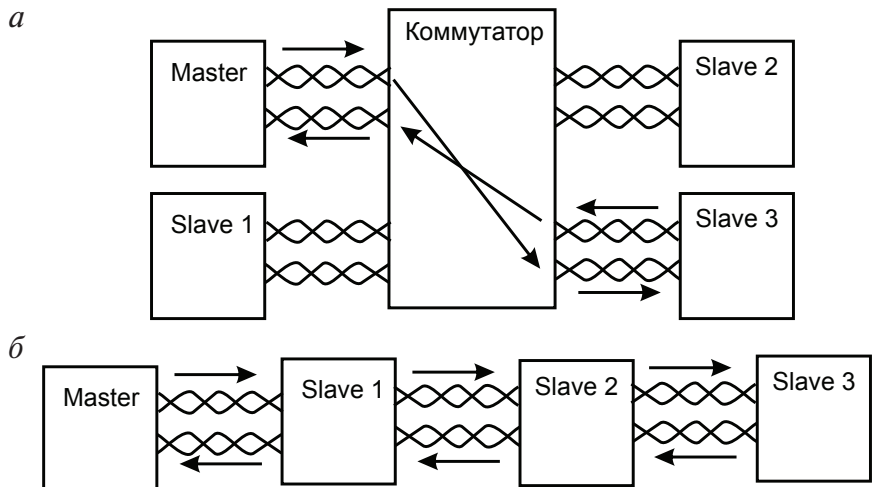


Рис. 21. Сравнение маршрута передачи информации в обычном сегменте сети Ethernet (*а*) и в сегменте, построенном по технологии EtherCAT (*б*)

Передача кадра без задержек на буферизацию и обработку позволяет оценить максимальное время оборота кадра в сети (время от момента начала его отправки ведущим устройством и до момента конца приема им же). Вследствие этого протоколом гарантируется доставка сообщения любому узлу в заранее определенный промежуток времени (время оборота кадра).

Формат кадра содержит в себе поля для всех имеющихся ведомых устройств в сети (рис. 22).

В связи с необходимостью модификации канального уровня для работы в сети EtherCAT требуются устройства с аппаратной поддержкой этой технологии. Для работы в качестве ведомого устройства такая поддержка обязательна. Ведущее устройство может использовать

стандартный сетевой интерфейс Ethernet, требуется только модифицированное ПО.

Рис. 22. Формат кадра EtherCAT

Технология EtherCAT описана для физической среды передачи, удовлетворяющей стандарту 100Base-TX (100 Мбит/с по витой паре). Разработка гигабитной версии протокола разработчиками пока не проводится и не планируется. Это может стать препятствием в развитии данной технологии при переходе на более скоростные сети Ethernet.

Контрольные вопросы

1. Какие недостатки технологии Ethernet затрудняют ее применение в сетях промышленной автоматизации?
2. Какие режимы обмена определены в протоколе PROFINET и в чем их отличие?
3. Какой алгоритм используется протоколом IEEE 1588 для синхронизации времени на устройствах в сети?
4. Как регламентируется доступ к среде передачи в протоколе POWERLINK?
5. Какие уровни стандарта Ethernet используются протоколом Ethernet/IP в качестве транспорта для передачи своих пакетов?
6. Какие уровни стандарта Ethernet модифицирует протокол EtherCAT?
7. В чем состоит отличие маршрута передачи кадра в сегменте сети EtherCAT от маршрута в обычном сегменте Ethernet?

6. Аппаратно-независимый протокол OPC

6.1. История развития

Создание стандарта OPC (OLE for Process Control) преследовало цель обеспечить разработчиков ПО верхнего уровня систем АСУТП (SCADA-системы, панели оператора, системы сбора и хранения информации о процессе) универсальным и открытым интерфейсом для взаимодействия непосредственно с системами управления технологическим процессом (ПЛК, интеллектуальные модули управления).

В 1990-х гг. доминирующей платформой для размещения ПО верхнего уровня была Microsoft Windows. Вследствие этого группой производителей автоматизирующего оборудования (Fisher-Rosemount, Intellution, Opto 22, Rockwell Software) в 1995–1996 гг. было принято решение о разработке единого стандарта для доступа к данным, базирующегося на механизме объектного связывания и внедрения (OLE), а именно на его реализации для компонентной объектной модели COM (DCOM — для взаимодействия через сети передачи данных). Эти базовые технологии были реализованы в ОС Windows. Результатом этой работы стал выпуск первой версии стандарта — OPC DA — OPC Data Access в августе 1996 г. Данная версия предназначалась для запроса текущих значений параметров технологического процесса. После этого еще несколько фирм включились в работу с новым стандартом, что привело в сентябре 1996 г. к учреждению Международной организации OPC Foundation, курирующей данный стандарт.

В 1999 г. была выпущена следующая редакция стандарта, получившая название OPC AE (Alarms & Events), которая включала поддержку передачи данных о событиях в работе системы управления. Дальнейшее развитие стандарт получил в 2001 г., когда вышла версия OPC

HDA (Historical Data Access), которая позволяла получить доступ к архиву значений, хранящемуся в памяти OPC-сервера. В 2003 г. вышла версия OPC XML-DA, описывающая обмен между клиентом и сервером посредством XML-языка запросов.

К 2003 г. монопольное положение Microsoft в нише HMI-систем сменилось конкуренцией нескольких платформ. Появились решения, базирующиеся на ОС Linux и других ОС с открытой архитектурой. В этих условиях привязка стандартов OPC к проприетарной технологии COM не удовлетворяла потребностям разработчиков ПО.

Дополнительными недостатками COM-технологии стали:

- закрытый компанией Microsoft исходный код, содержащий ошибки;
- ненастраиваемые времена ожидания при взаимодействии по сети;
- низкая безопасность при сетевом обмене данными.

Вследствие этого консорциумом OPC Foundation в 2006 г. была выпушена спецификация принципиально нового стандарта OPC UA (Unified Architecture). Сама аббревиатура OPC тоже стала расшифровываться по-другому — Open Platform Communication. Предыдущие спецификации были объединены под одним именем — OPC Classic. В стандарте OPC UA обмен выполняется посредством одного из двух механизмов: 1) протокол HTTP (протокол 7-го уровня по модели МВОС); 2) путем непосредственного использования протокола TCP и нижележащих протоколов (уровни с 1-до 4-го по модели МВОС). Полностью описанный в стандарте стек протоколов OPC UA позволяет создавать ПО с открытым исходным кодом (например, на языках C++, Java, C#), которое легко может быть экспортировано на любую из имеющихся программно-аппаратных платформ без ущерба в функциональности.

Новый стандарт также обладает рядом преимуществ:

- кроссплатформенность;
- повышенная возможность масштабирования приложений;
- безопасность коммуникаций;
- конфигурируемые времена задержек при обмене;
- многопоточность.

В 2012 г. стандарт был оформлен в рамках Единого реестра европейских стандартов под номером IEC 62541. В настоящее время к консорциуму присоединились более 480 участников из разных стран мира [15].

В основе технологии OPC лежит взаимодействие по принципу клиент-сервер. Сервер OPC представляет собой программу на устройстве, располагающем данными о технологическом процессе. Это может быть как непосредственно устройство уровня управления (ПЛК, интеллектуальный модуль и т. п.), так и устройство уровня человеко-машинного интерфейса (SCADA-система, выступающая в этом случае промежуточным звеном между ПЛК и OPC-клиентом). OPC-сервер ожидает подключения клиентов, после установления подключения сервер принимает от клиента запрос данных, после чего формирует и отправляет ответ на него.

Клиентом OPC является программа, которой необходимо получить данные о процессе. Как правило, это SCADA-система, выполняющая свои основные функции (визуализация данных, архивация значений параметров, оповещение о событиях в работе системы).

Взаимодействие клиента и сервера может быть как локальным (когда обе программы запущены на одном компьютере), так и удаленным посредством имеющейся инфраструктуры передачи данных. В качестве сети передачи данных в стандарте OPC используется локальная сеть, построенная по технологии Ethernet.

В настоящее время можно выделить несколько стандартов из группы OPC:

- OPC DA — исторически первый появившийся и наиболее часто используемый в настоящее время стандарт. Если какое-либо устройство заявлено как имеющее поддержку OPC, это значит, что оно, как минимум, поддерживает стандарт OPC DA. Данный стандарт позволяет считывать текущие параметры технологического процесса (значения, время измерения, качество);
- OPC AE — описывает обмен сообщениями о нештатных ситуациях в работе системы, а также о различных состояниях, переключениях и подобном;
- OPC HDA — описывает методы запросов к архиву данных, хранящемуся на OPC-сервере;
- OPC UA — описана современная архитектура взаимодействия с помощью механизмов HTTP-запросов, а также путем формирования TCP-пакетов собственного формата.

6.2. Стандарт OPC DA

Стандарт основан на технологии межпроцессного взаимодействия COM/DCOM и описывает перечень COM-объектов, которые в совокупности позволяют получить требуемые данные об объектах автоматизации. Описанные объекты реализуются в программе, которая выполняет в этом случае функцию OPC-сервера.

OPC-сервер может быть как встроен в устройство управления (ПЛК), так и реализован программно в качестве вспомогательного приложения, входящего в пакет компьютерных программ для взаимодействия с оборудованием.

В первом случае OPC-сервер является неотъемлемой частью автоматизирующего оборудования (рис. 23, *а*). Однако, т. к. для функционирования сервера необходимо функционирование ОС, поддерживающей технологию COM, этот вариант практически никогда не используется для данного стандарта. Это связано с тем, что поддержка технологии COM в достаточной степени реализована только в ОС компании Microsoft, которые не получили широкого распространения как встраиваемые ОС для контроллеров, управляющих технологическим процессом. В этом качестве, как правило, используются либо встроенные операционные системы конкретного производителя, либо системы на базе свободно распространяемой ОС Linux (TinyOS).

Если OPC-сервер реализован как программа на компьютере (рис. 23, *б*), то он, как правило, является частью SCADA-системы, предоставляемой производителем данного оборудования, и позволяет получать данные с этого оборудования сторонним программам-клиентам (например, общецеховым и общезаводским системам анализа, хранения и архивации данных). Запущенная SCADA-система в этом случае взаимодействует с конкретным ПЛК по специфичному для него протоколу (например, SCADA-система WinCC компании Siemens взаимодействует с ПЛК S7–1200 (1500) этого же производителя по протоколу S7-Automation). С другой стороны, данное ПО предоставляет функционал OPC-сервера и позволяет подключить по открытому протоколу OPC один или несколько сторонних клиентов для доступа к данным о технологическом процессе.

Рис. 23. Схемы использования стандарта OPC в схемах АСУТП:
а) OPC-сервер встроен в ПЛК, *б*) OPC-сервер как программа на компьютере

Задача OPC-клиента состоит в правильном взаимодействии с определенными в стандарте интерфейсами COM-объектов сервера. При этом, так как спецификация COM определяет обмен на уровне байт данных, разработка клиентов для OPC-подключения может быть выполнена на любом языке программирования, поддерживающем работу с COM (C++, C#, VB, Java и др.).

6.2.1. Описание технологии COM применительно к стандарту OPC

Технология COM была разработана компанией Microsoft в 1993 г. для организации взаимодействия между процессами (межпроцессное взаимодействие). Так как каждый процесс работает в своем локальном адресном пространстве, то в случае, когда требуется передать данные от одного процесса другому, требуется вмешательство ОС и наличие механизмов, позволяющих это сделать. Простейшими механизмами передачи данных между процессами являются общие файлы и буфер обмена. Однако эти способы взаимодействия недостаточно надежны и безопасны.

Ключевыми сущностями в данной парадигме являются COM-компоненты, к которым другой процесс может обращаться посредством COM-интерфейсов (наборов абстрактных свойств и функ-

ций, реализация которых пишется разработчиком COM-компонента и скрыта от клиента).

Компоненты можно сравнить с классами языка C++ с той разницей, что класс доступен только в том же адресном пространстве, где расположена вызывающая его программа, а COM-компонент функционирует в отдельном адресном пространстве, где могут выполняться другие задачи (например, в случае SCADA-системы — опрос оборудования и визуализация). Другим отличием является то, что класс имеет только один интерфейс, а COM-объект может иметь много интерфейсов.

COM-интерфейсы, как и классы C++, могут быть реализованы и как часть EXE-файла, и как часть DLL-библиотеки. В случае реализации в DLL-библиотеке ОС при обращении к компоненту создает процесс с программой-заглушкой, в адресное пространство которого и загружается компонент из библиотеки.

Для функционирования клиента и сервера OPC организацией OPC Foundation был выпущен набор библиотек, описывающих COM-объекты для различных стандартов OPC. Общее название этого набора OPC Core Components SDK (табл. 18).

Таблица 18

Наиболее часто используемые библиотеки из OPC Core Components SDK

Название файла библиотеки	Стандарт	Последняя версия
opchda_ps.dll	OPC Historial Data Access	1.20
opc_aeps.dll	OPC Alarms & Events	1.10
opcproxy.dll	OPC Data Access	3.0

В дальнейшем будем называть обращающуюся к COM-компоненту программу *клиентом*. Возможны три случая обращения клиента к COM-компоненту:

- внутризадачное размещение — при обращении клиента COM-компонент загружается как DLL-библиотека в адресное пространство клиента («внутрь» его задачи) и выполняется в нем. Этот способ практически сводит на нет все преимущества COM-компонента по сравнению с обычной библиотекой функций и поэтому используется достаточно редко;
- локальное размещение — COM-компонент размещается в отдельном адресном пространстве того же компьютера, на кото-

ром запущен клиент. Этот способ используется, если OPC-сервер и OPC-клиент расположены на одном компьютере;
- удаленное размещение — COM-компонент запущен на отдельном компьютере, к которому по сети обращается клиент. Этот способ используется, если OPC-сервер и OPC-клиент расположены на разных компьютерах.

Последние два способа размещения COM-компонента используют возможности ОС (а именно ее технологию RPC) для передачи информации через границу локального адресного пространства клиента.

Дополнительным преимуществом COM-компонента является то, что клиенту нет необходимости знать, где физически расположен исполняемый файл компонента. Так как ОС всегда является посредником между клиентом и сервером, достаточно хранить в средствах ОС информацию о расположении всех компонентов. Когда клиенту потребуется эта информация, ОС выдаст ее. Передача и сохранение в операционной системе информации о COM-компонентах происходит в момент установки программы, содержащей компонент, и называется регистрацией компонентов в системе. Для зарегистрированных компонентов Windows хранит в реестре уникальный номер компонента (CLSID), место расположения файла с его кодом, краткое описание и другую служебную информацию. Клиент обращается к COM-компоненту через вызов функции операционной системы. При этом вызове ОС по информации из реестра определяет процесс, управляющий компонентом (если он не запущен, то запускает его), и передает ссылку на интерфейс компонента клиенту. Клиент, пользуясь предоставленной ссылкой, может запускать методы интерфейса, считывать значения свойств и т. п. Общая схема работы с COM-компонентом изображена на рис. 24.

6.2.2. Структура OPC-сервера

Существуют спецификации стандарта для компилируемых (Custom Interface Standard) [16] и для интерпретируемых (Automation Interface Standard) [17] языков. (рис. 25).

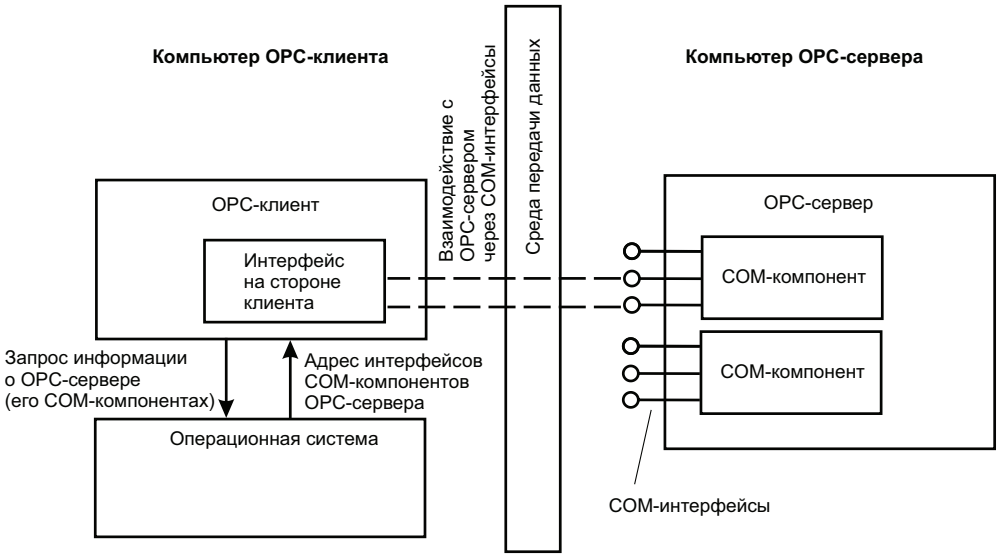


Рис. 24. Схема взаимодействия компьютеров клиента и сервера по протоколу OPC

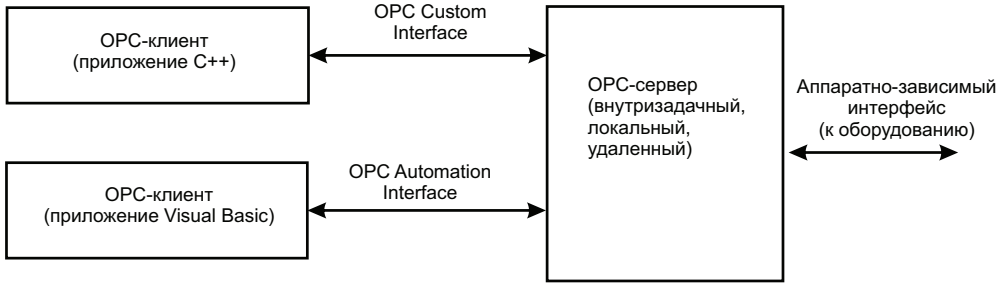


Рис. 25. OPC-интерфейсы

Интерфейс Automation появился, начиная с версии 2.0 спецификации, в связи с тем, что COM-объекты изначально не были спроектированы для работы с интерпретируемыми языками. Для решения этой задачи используется вызов промежуточных функций OPC Automation Wrapper (рис. 26).

В стандарте OPC-DA Custom 2.05 [16] описаны следующие COM-объекты, обеспечивающие функционирование OPC-сервера:

- OPCServer Object;
- OPCGroup Object.

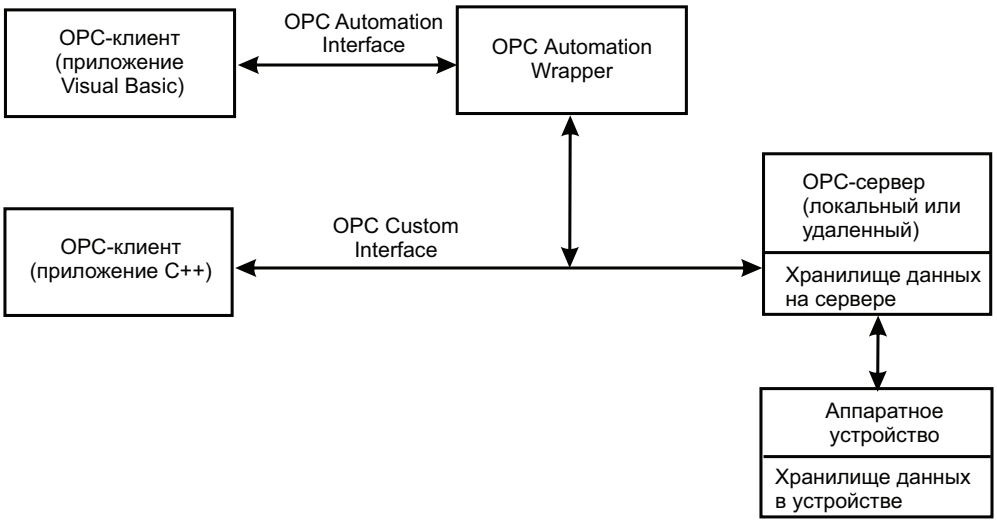


Рис. 26. Схема взаимодействия с сервером OPC различными клиентами

OPCServer Object

Данный объект служит для получения данных от устройства-источника (ПЛК). OPC-клиенты подключаются к данному объекту для запроса этих данных.

Объект **OPCServer** обладает следующими стандартизованными интерфейсами:

- IUnknown (стандартный интерфейс всех COM-объектов);
- IOPCCommon — вспомогательный интерфейс (для установки используемого языкового стандарта при взаимодействии с клиентом, получения текстовых сообщений об ошибках и. т.п);
- IOPCServer — основной интерфейс при взаимодействии с сервером: работа с группами (создание/перебор/удаление), работа с точками подключения;
- IOPCServerPublicGroups (необязательный) — интерфейс для работы с общими (публичными группами), т. е. группами, доступными нескольким клиентам;
- IOPCBrowseServerAddressSpace (необязательный) — интерфейс для просмотра адресного пространства сервера (линейного или иерархического);
- IPersistFile (необязательный) — интерфейс служит сохранению или загрузке в сервер конфигурации из файла. Позволяет клиенту удаленно воздействовать на конфигурацию сервера;

- IOPCItemProperties — используется для просмотра свойств объекта сервера;
- IConnectionPointContainer — обеспечивает работу с точками подключения.

Адресное пространство сервера может быть организовано произвольным образом и не коррелировать с разбиением на группы:

- фиксированное адресное пространство;
- конфигурируемое динамически с момент старта сервера;
- конфигурируемое на лету в зависимости от данных, запрашиваемых клиентами в каждый момент времени.

Как правило, адресное пространство сервера организовано иерархически:

Сервер

- Группа (Group) 1:
  — Элемент (Item) 1.1;
  — Элемент (Item) 1.2;
- Группа (Group) 2:
  — Элемент (Item) 2.1

У сервера могут быть определены так называемые общие группы. Такая группа создается одним клиентом, но может использоваться несколькими клиентами, которым нужны те же данные, что и создавшему ее клиенту. Так как предполагается одновременный доступ к одним и тем же данным, в общих группах предусмотрены механизмы упорядочения доступа.

OPCGroup Object

Для удобства манипулирования данными клиентом, данные, получаемые от сервера, могут быть объединены в группы. Группа управляется как неделимый элемент. Следует отметить, что объединение данных в группы выполняется клиентом, а не сервером, с целью удобного доступа к информации (запрос всех данных группы, подписка на группу и т. д.). Поэтому разбивка данных на группы клиентозависима и не оказывает влияния на состояние сервера OPC.

Объект **OPCGroup** обладает следующими стандартизованными интерфейсами (по версии спецификации V2.05):

- IUnknown (стандартный интерфейс всех COM-объектов);
- IOPCItemMgt — позволяет добавлять и удалять элементы в группу, управлять элементами в группе;

- IOPCGroupStateMgt –используется для получения свойств группы;
- IOPCPublicGroupStateMgt (необязательный) — если группа создана как общая, то данный интерфейс используется для определения ее состояния, что важно при доступе нескольких клиентов;
- IOPCSyncIO — получить данные в синхронном режиме;
- IOPCASyncIO2 — получить данные в асинхронном режиме;
- IConnectionPointContainer– обеспечивает работу с точками подключения.

Объект **OPCGroup** также обладает рядом свойств, влияющих на его поведение:

- имя — должно быть уникальным среди всех групп данного клиента. У общих групп свое пространство имен, в котором каждое имя также должно быть уникальным;
- кэшированные данные (показатель, где находятся данные группы — в хранилище OPC-сервера или в хранилище устройства);
- статус активности. Показывает, надо ли серверу обновлять данные из этой группы или нет (важен для экономии ресурсов сервера);
- скорость обновления данных (Update Rate);
- временная зона (используется, когда клиент и сервер в разных часовых поясах);
- ширина мертвой зоны (используется при обновлении аналоговых значений для экономии ресурсов сервера);
- идентификатор клиента (ClientHandle) — нужен для идентификации группы клиентом;
- чтение и запись данных — чтение может быть организовано тремя способами (1 — синхронный, 2 — асинхронный, 3 — по подписке, т. е. по изменению), запись — двумя (синхронно или асинхронно);
- для публичных групп свойства таких групп должны сохраняться сервером для каждого клиента отдельно.

OPC-клиент также может создать ряд интерфейсов для взаимодействия с сервером.

IOPCDataCallBack — интерфейс для подключения сервера к клиенту с помощью точек подключения.

6.2.3. Алгоритм взаимодействия клиента с сервером

Тонкости алгоритма взаимодействия клиента и OPC-сервера, работающих по технологии OPC DA, подробно изложены в работе Федоренко Д. Ю. [6].

Получение списка серверов

В начале работы клиент должен получить список имеющихся локальных OPC-серверов. Для этого используется предоставляемая бесплатно OPC Foundation библиотека типов OPCEnum. Данную библиотеку можно установить самостоятельно. Также она, как правило, устанавливается в систему при установке одной из SCADA-систем. Версия данной библиотеки для платформы.NET позволяет напрямую подключать ее в свой проект на языках CLR (C#, VB) и использовать имеющиеся в ней типы и функции.

Получение адресного пространства сервера

Для получения адресного пространства сервера, как правило, используется необязательный, но часто реализуемый в серверах интерфейс IOPCBrowseServerAddressSpace. Он позволяет получить развернутое адресное пространство в виде иерархического дерева.

Создание групп параметров для опроса сервера

Клиент вызывает метод AddGroup интерфейса IOPCServer. После этого во вновь созданную группу добавляются элементы с помощью функции AddItems интерфейса IOPCItemMgt.

Существуют функции синхронного и асинхронного чтения данных с сервера. Функция синхронного чтения блокирует поток, из которого она была вызвана, до завершения операции чтения. Когда операция чтения завершится, выполнение основного потока будет продолжено далее. Асинхронное чтение не блокирует вызывающий поток.

Синхронное чтение

Выполняется вызовом метода Read интерфейса IOPCSyncIO. Результат чтения помещается в массив структур OPCITEMSTATE.

Асинхронное чтение

При асинхронном чтении данные считываются на стороне клиента в отдельном потоке. Для запуска такого потока клиент должен скон-

фигурировать сервер таким образом, чтобы он сам запустил передачу данных, вызвав функцию на стороне клиента. Данная функция должна быть реализована клиентом как COM-интерфейс, определенный в стандарте OPC и называющийся IOPCDataCallback. Он должен быть реализован на стороне клиента.

Для того чтобы передать серверу ссылку на интерфейс клиента, клиент обращается к интерфейсу сервера IOPCConnectionPointContainer. Далее через этот интерфейс необходимо запросить точку подключения с помощью функции FindConnectionPoint, передав ей в качестве аргумента идентификатор интерфейса клиента IOPCDataCallback (это нужно, чтобы сервер смог проанализировать, сможет ли он воспользоваться данным интерфейсом). Если функция завершается успешно, она возвращает созданную сервером точку подключения (интерфейс IConnectionPoint). Для активирования полученной точки подключения клиент вызывает ее метод Advice, который возвращает идентификатор подписки. Этот идентификатор может быть использован в дельнейшем для отмены подписки.

Команда для сервера начать асинхронную передачу данных выполняется на стороне клиента вызовом метода Read интерфейса IOPCAsync2. Сервер выполнит передачу данных и вызовет метод интерфейса клиента IOPCDataCallback с именем OnReadComplete.

Чтение данных по изменению

В некоторых случаях целесообразно предавать данные от сервера к клиенту только в случае их изменения. Особенно это актуально для дискретных величин (битовых или целых). Такая стратегия позволяет разгрузить обмен данными между сервером и клиентом от передачи повторяющихся неизменных значений.

Алгоритм передачи данных по изменению аналогичен алгоритму асинхронного чтения: клиент делает запрос на создание точки подключения, после чего вызывает метод Advice для организации подписки. При изменении данных на стороне сервера будет вызвана функция OnDataChange на стороне клиента, которой от сервера будет передан массив измененных данных. Вызывать метод IOPCAsync2: Read в этом случае не требуется.

6.2.4. Особенности реализации для платформы .NET

Так как технология.NET появилась позже, чем инфраструктура COM, то использовать библиотеки COM-объектов в проектах, написанных на управляемых языках (C#, VB), напрямую нельзя. Однако компанией Microsoft был разработан механизм трансляции между управляемым кодом и объектами COM, который получил название Runtime Callable Wrapper. В его задачи входит преобразование типов данных COM в типы, совместимые в платформой .NET. Например, тип, объявленный в библиотеке COM как HRESULT, будет преобразован в языке C# в тип System.UInt32.

Кроме этого, преобразуются вызовы функций, а также выполняется безопасная передача объектов, созданных в неуправляемой памяти.

Однако данный механизм трансляции не всегда может преобразовать некоторые интерфейсы COM-объектов корректно, поэтому требуется быть внимательным при его использовании.

В свою очередь в OPC Foundation была разработана библиотека функций для работы с OPC в платформе.NET. Эта библиотека называется OPC. NET API и распространяется в составе OPC Core Components. В отличие от RCW данная библиотека спроектирована для работы непосредственно с технологией OPC, что позволяет ей избежать ошибок, свойственных автоматической трансляции, используемой в RCW. Однако, используя данную библиотеку, пользователь ставится в зависимость от разработчика и при изменении спецификаций не сможет быстро и самостоятельно адаптировать свое ПО под них (будет вынужден ждать выхода новой версии библиотеки OPC. NET API).

6.3. Стандарт OPC UA

6.3.1. Общие положения

Стандарт OPC Unified Architecture (UA) [18] был разработан организацией OPC Foundation как замена группе стандартов OPC, в основе которых лежала проприетарная технология COM. Предполагается, что данный стандарт найдет применение во многих областях промышленности: системы управления, включая датчики и исполнительные ме-

ханизмы, интернет вещей, взаимодействие «машина-машина» (M2M). Стандарт описывает как общую модель взаимодействия таких систем, так и частности:

- информационную модель, включающую структуры, поведение и семантику;
- модель обмена сообщениями для взаимодействия между приложениями;
- модель совместимости для обеспечения взаимодействия между разными подсистемами.

OPC UA является платформонезависимым стандартом, описывающим взаимодействие с помощью сообщений между клиентом (client) и сервером (server), а также между подписчиком (subscriber) и издателем (publisher) в случае взаимодействия с помощью подписки.

Клиент-серверная модель описывает набор сервисов (service), который предоставляет сервер для получения данных клиентом, включая доступ к актуальным данным, архиву, набору сообщений об авариях и событиях, т. е. данный стандарт включает в себя функционал нескольких предшествующих стандартов.

В дополнение к клиент-серверной модели стандарт описывает взаимодействие по модели подписки, когда сервер оповещает клиента о наступлении каких-то событий.

Все спецификации данного стандарта разделены на уровни в соответствии с рис. 27.

Уровни форматирования, защищенного канала и транспортный объединены общим названием — стек OPC UA. Необходимо отметить, что данные уровни не совпадают с уровнями по модели МВОС. По данной модели все уровни стека OPC UA относятся к прикладному уровню, а в качестве протоколов линии передачи подразумеваются протоколы уровней с четвертого (TCP) по седьмой (HTTPS) модели МВОС.

Такое абстрагирование позволяет реализовать функционал OPC разработчику прикладного ПО (приложения OPC UA), оставив реализацию нижележащих сетевых уровней имеющимся технологиям локальных сетей передачи данных.

Транспортный уровень стека OPC UA состоит из протокола OPC UA Connection Protocol, работающего поверх одного из протоколов для передачи:

- OPC UA TCP (работает поверх стандартного протокола TCP);

- HTTPS (работает поверх протокола HTTPS);
- WebSockets (работает поверх протокола межсерверного соединения WebSockets).

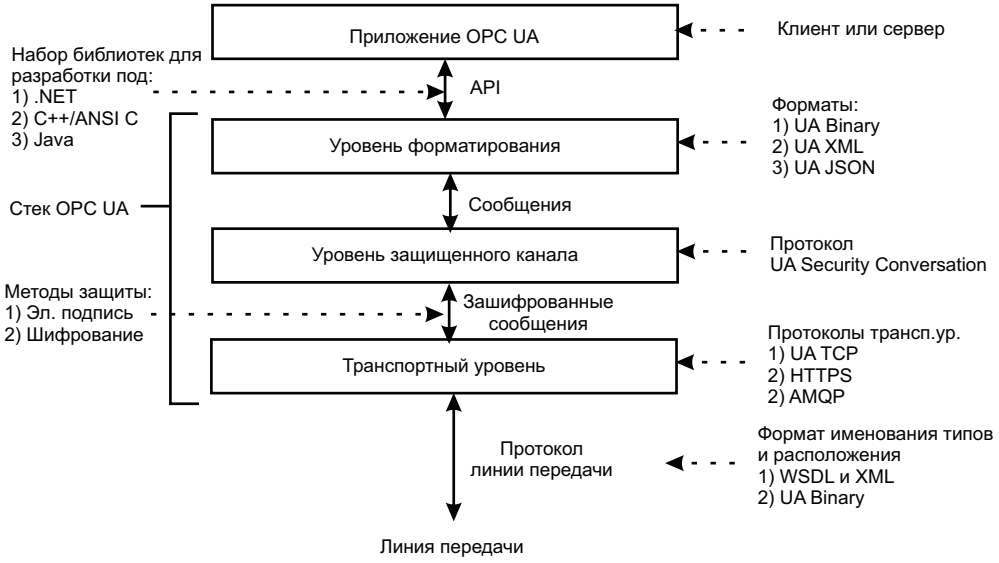


Рис. 27. Обзор уровней стека OPC UA

Для того чтобы подключиться к серверу клиент использует его точки подключения. Точка подключения содержит набор сервисов, которые доступны клиенту, если он подключится к ней.

Точка подключения содержит уникальный для сервера набор следующих характеристик:

- Имя (EndpointUrl);
- Режим безопасности (SecurityMode);
- Используемый алгоритм авторизации и шифрования (SecurityPolicyUri).

Имя каждой точки подключения однозначно определяет, какой транспортный протокол должен использовать клиент, а также TCP-порт в случае использования TCP-протокола. Для каждого из протоколов определен свой формат имени точек подключения при обращении к серверу.

Пример имени точки подключения:

opc.tcp:\\NameServ:4840

где opc.tcp — часть адреса, указывающая, что сервер поддерживает протокол OPC UA TCP; NameServ — сетевое (IP-адрес), или доменное имя сервера; 4840 — номер порта TCP для подключения к серверу.

Аналогичные примеры для протоколов HTTPS и WebSockets:

https:\\NameServ:443;
opc.wss:\\NameServ:443.

Режим безопасности (SecurityMode) — определяет параметры безопасности при использовании данной точки подключения: None — не используются механизмы безопасности; Sign — используется сертификат с цифровой подписью для установления соединения; SignAndEncrypt — используется сертификат с цифровой подписью для установления соединения и шифрование в процессе обмена.

Примеры алгоритмов авторизации и шифрования (SecurityPolicyUri):

None — используется в случае соответствующего режима безопасности None.

Aes128-Sha256-RsaOaep — для обмена информацией используется симметричное шифрование алгоритмом AES-256. Для установления соединения и формирования сессионного ключа используется ассиметричное шифрование RSA с функцией хэширования SHA-256.

Если прибавить к этим трем атрибутам имя сервера, то совокупный набор делает каждую точку подключения уникальной в списке всех точек подключения на данном компьютере.

Каждый сервер должен иметь точку подключения DiscoveryEndpoint, не требующую установления сеанса (режим безопасности — None, используемый алгоритм шифрования — None). В списке точек подключения сервера данная точка всегда идет первой. Клиент, подключившись к данной точке, может с помощью функции GetEndpoints () считать информацию о параметрах всех других точек подключения данного сервера. После этого он обращается к одной из найденных точек для установления защищенного канала и формирования сеанса.

Список существующих на узле серверов может быть получен клиентом, если на узле работает специальный LocalDiscovery-сервер с известным именем:

opc.tcp:\\localhost:4840.

Подключившись к Discovery-серверу клиент с помощью функции FindServers () получает список OPC-серверов и может обращаться к ним.

Уровень защищенного канала организует защищенный канал передачи данных, учитывающий привилегии клиента.

Уровень форматирования описывает форматы сообщений, которыми обмениваются клиент и сервер. Стандартом описаны три способа форматирования информации:

- UA Binary;
- UA XML;
- UA JSON.

Приложение (OPC клиент или сервер) обращается к стеку OPC с помощью набора библиотек функций. Необходимо отметить, что функции и классы из этих библиотек не являются частью спецификации OPC и зависят от используемой платформы разработки (.NET, Java, x86). Тем не менее, организацией OPC Foundation были разработаны наборы библиотек для использования OPC под основные современные платформы и языки (C++/ANSI C, C#, Java).

Благодаря наличию нескольких вариантов протоколов и способов кодирования, разработчик OPC UA приложений может выбирать (или дать выбрать пользователю) один из имеющихся вариантов для максимального удобства (например, выбрать более быстрый протокол, но требующий больших усилий для программирования обмена).

Модель «издатель-подписчик» (PubSub) позволяет организовать обмен в следующих случаях:

- когда нужно организовать обмен данными между контроллерами или между контроллером и HMI-устройством. При этом участники обмена не связаны непосредственно и не требуют наличия партнера по обмену в каждый момент времени (если партнера нет в сети, это не приводит к негативным последствиям для оставшегося в сети узла);
- асинхронный поток данных. Например, в случае, когда одно приложение размещает свои заказы в одной очереди сообщений, откуда они могут быть выбраны одним или несколькими исполнителями;
- сохранение данных в нескольких местах. Например, передача данных от датчика в систему мониторинга, HMI-систему и в приложение для архивации одновременно;

- серверы, предоставляющие данные для приложений, находящихся в облаке. Например, приложения для аналитики, оптимизации и предсказания (когда неизвестно заранее, кто и когда запросит эти данные).

6.3.2. Безопасность обмена

Механизмы аутентификации обмена данными между клиентом и сервером описаны в части 2 стандарта OPC UA [19]. Там определены так называемые профили (security profiles), которые должны поддерживать все приложения, взаимодействующие по этому протоколу. В начале сессии обмена данными организуется защищенный канал связи между клиентом и сервером.

В основе шифрования лежит использование стандартных сертификатов X.509 на стороне сервера. Каждый OPC-сервер должен обладать таким сертификатом и предоставлять его клиенту. Клиент отправляет зашифрованное публичным ключом сервера сообщение OpenSecureChannel, в котором содержится секретный ключ сеанса. Кроме этого, в данном сообщении серверу передается открытый ключ клиента. Сообщение подписывается секретным ключом клиента. Сервер расшифровывает данное сообщение, проверяя подпись с помощью открытого ключа клиента. Весь дальнейший обмен сообщениями происходит с помощью симметричного шифрования имеющимся у каждой стороны сеансовым ключом.

При взаимодействии по модели подписки набор криптографических ключей должен быть передан перед созданием подписки с помощью установления обмена по методу клиент-сервер (см. выше).

Так, аутентификация участников обмена выполняется однократно — в момент начала установления соединения. Для взаимной идентификации клиент и сервер, как правило, используют цифровые сертификаты стандарта X.509 (асинхронное шифрование сессионного ключа методом RSA и последующая передача данных, зашифрованных синхронным шифром).

При этом необходимо, чтобы каждая из сторон располагала своим сертификатом и принимала сертификат другой стороны.

Сервер, как правило, представляет собой коммерческую разработку и располагает сертификатом, подписанным одним из международ-

ных центров сертификации. Клиент может также располагать таким сертификатом или, в случае, когда разработка клиента выполняется «с нуля», может создать свой самоподписанный сертификат.

Проблема настройки доверия между клиентом и сервером сводится к признанию каждой стороной сертификата другой стороны. Если сертификат подписан одним из центров сертификации и может быть проверен, то сторона, владеющая им, признается заслуживающей доверия, и с ней может быть установлена сессия для обмена данными. Проблемы возникают, как правило, с клиентским самоподписанным сертификатом. В этом случае необходимо указать серверу вручную, что данному сертификату можно доверять. Например, для OPC UA-сервера компании Siemens, который входит в комплект программ Simatic NET, эта настройка выглядит так, как показано на рис. 28.

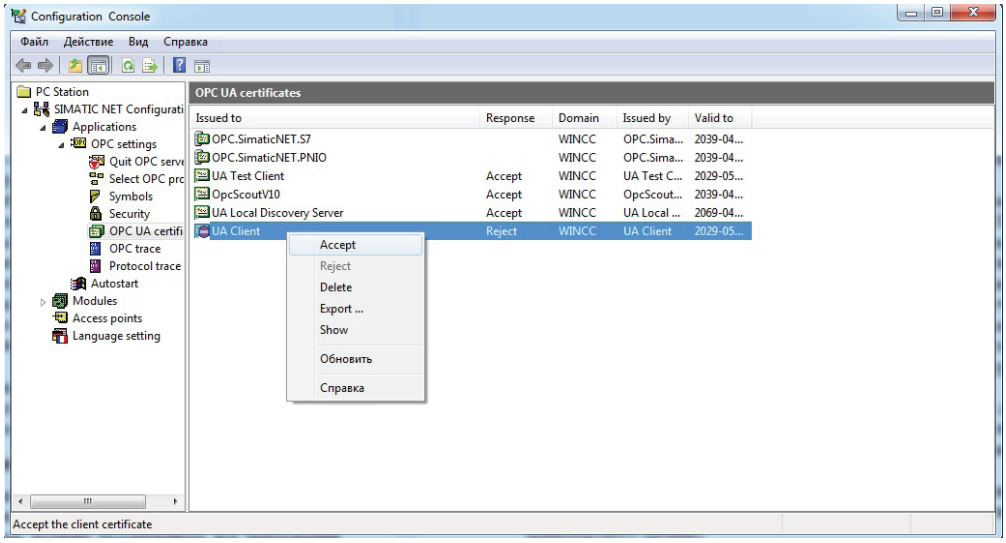


Рис. 28. Внесение сертификата UA Client в доверенные сертификаты с помощью настроек Simatic NET Configuration Console

Имеющиеся на компьютере сертификаты могут храниться в хранилище сертификатов операционной системы или отдельно в директориях. Хранение в хранилище ОС более предпочтительно, т. к. в этом случае сертификаты защищаются операционной системой и могут управляться любыми программами, установленными на данном компьютере. Для помещения сертификата в хранилище ОС программа должна быть запущена с правами администратора.

Просмотр содержимого хранилища сертификатов выполняется с помощью оснастки mmc. Необходимо в меню Выполнить... набрать mmc и нажать Enter. В появившемся окне необходимо раскрыть пункт меню Файл и выбрать подпункт Добавить или удалить оснастку... После этого нужно выбрать среди доступных оснасток Сертификаты и нажать кнопку Добавить (рис. 29).

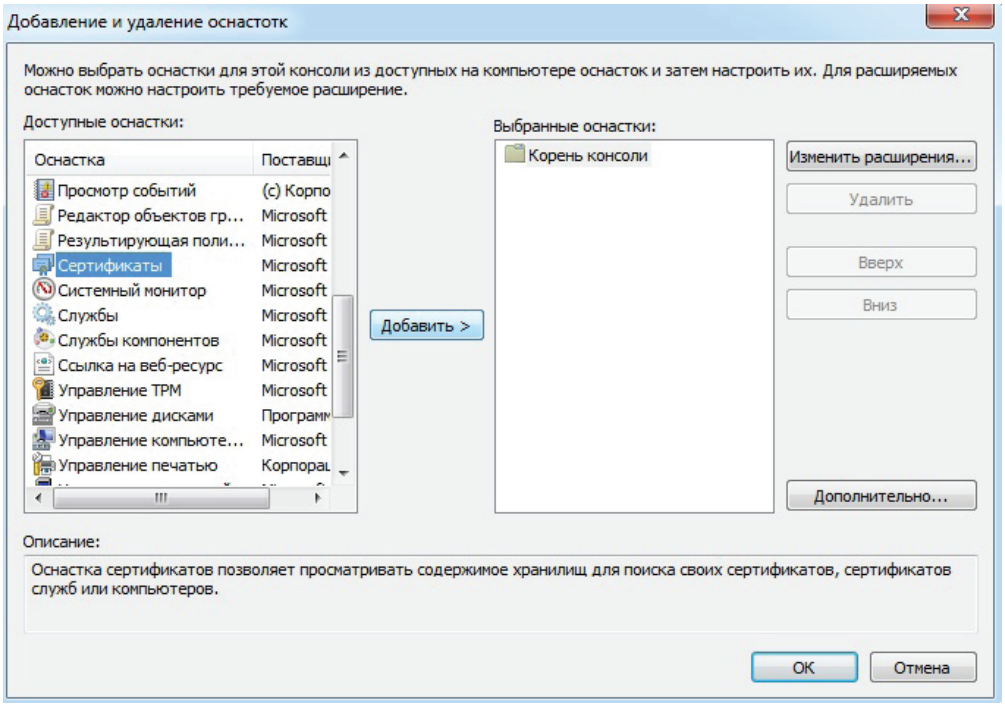


Рис. 29. Добавление в консоль оснастки управления сертификатами

Далее нужно выбрать переключатель «учетной записи компьютера», чтобы увидеть сертификаты, привязанные к данному компьютеру, и нажать кнопку OK. Добавленная оснастка позволяет просматривать сертификаты, привязанные к компьютеру (рис. 30).

Сертификаты центров сертификации помещаются в группу Доверенные корневые центры сертификации. Сертификаты, внесенные вручную, помещаются в подхранилище Личное (My в англ. варианте Windows). Использование этих сертификатов может потребовать дополнительных настроек для признания их безопасными некоторыми приложениями (например, OPC UA-сервером).

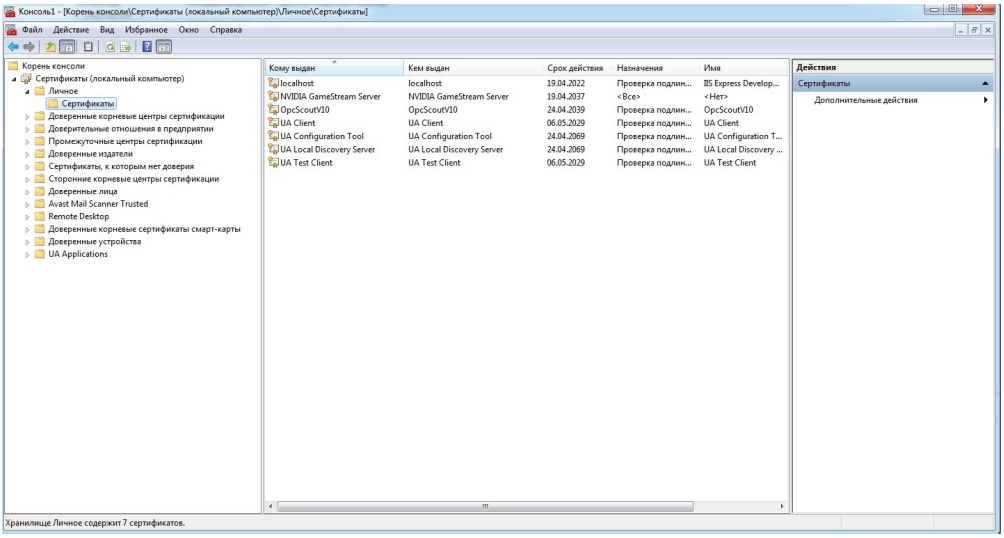


Рис. 30. Окно просмотра хранилища сертификатов

Сообщения системы безопасности могут сохраняться в долговременном хранилище (audit log). Такие сообщения может генерировать, например, сервер при попытке доступа к данным с ненадлежащим уровнем привилегий клиента.

Также OPC UA может использовать для шифрования возможности нижележащих сетевых протоколов (как правило, возможности протокола транспортного уровня).

6.3.3. Модель адресного пространства

Сервер описывает для клиента объекты автоматизации посредством своего адресного пространства [20]. Модель объекта представлена как набор переменных, значения которых можно считать, записать или получить уведомление о их изменении, а также методов, которые клиент может вызывать и получать уведомления после завершения их работы.

Совокупность объектов, которые сервер предоставляет клиенту, называется адресным пространством (address space). Адресное пространство представлено как совокупность узлов (nodes), соединенных связями (references).

Узел состоит:

- из атрибутов. Для каждого узла атрибуты имеют свои конкретные значения;
- ссылок на другие узлы.

Атрибут состоит из полей:

- id (идентификационного номера);
- имени;
- описания;
- типа данных;
- индикатора «обязательный» или «необязательный».

Связь между узлами состоит из полей:

- узел-источник;
- узел назначения;
- тип связи.

Узел назначения может находиться в адресном пространстве как того же, так и другого OPC UA-сервера.

Узлы могут различаться набором атрибутов в зависимости от их назначения. Для определения узлов используется терминология классов объектно-ориентированного программирования. Все типы узлов определены в части стандарта, которая называется классы узлов (NodeClasses). Ни клиент, ни сервер не могут создавать свои классы и должны использовать те, что определены в стандарте.

Базовый класс (Base NodeClass) включает в себя общие для всех классов узлов атрибуты. Остальные описанные классы наследуются от базового и могут содержать свои собственные атрибуты.

Состав атрибутов базового класса приведен в табл. 19.

Таблица 19

Список атрибутов узла класса BaseNodeClass

Атрибут	Обязательный (M)/ Необязательный (O)	Тип данных
NodeId	M	NodeId
NodeClass	M	NodeClass
BrowseName	M	QualifiedName
DisplayName	M	LocalizedText
Description	O	LocalizedText
WriteMask	O	AttributeWriteMask
UserWriteMask	O	AttributeWriteMask
RolePermissions	O	RolePermissionType []
UserRolePermission	O	RolePermissionType []
AccessRestriction	O	AccessRestrictionType

NodeId — структура, определяющая уникальный идентификатор в пределах сервера.

NodeClass — атрибут, определяющий класс узла.

BrowseName — имя, отображаемое посредством сервиса обзора адресного пространства сервера. Разные узлы могут иметь одинаковые BrowseName. Оно состоит из пространства имен (namespace) и строкового имени. Пространство имен может совпадать с таковым, указанным в структуре NodeId, или не совпадать. Строковое имя чувствительно к регистру.

DisplayName — имя, отображаемое для пользователя. Клиент OPC должен использовать это имя для предъявления его пользователю. Максимальная длина имени — 512 символов.

Description — текстовое описание.

WriteMask — определяет, в какие атрибуты узла клиент может записать новое значение (записываемые атрибуты). Если сервер не располагает этой информацией относительно некоторых атрибутов, он должен помечать их как записываемые. Если происходит запись в атрибут, сервер пробует выполнить эту операцию, и возвращает клиенту код статуса операции (StatusCode).

UserWriteMask — определяет, в какие атрибуты узла клиент может записать новое значение, учитывая права текущего аккаунта пользователя. Этот атрибут ужесточает разрешения, указываемые в WriteMask.

RolePermissions — определяет параметры доступа к узлу для всех ролей, которые имеют к нему доступ.

AccessRestriction — атрибут, показывающий дополнительное ограничение доступа к узлу

Интерес представляет также класс Variable, т. к. через узлы этого класса, как правило, передаются данные от клиента к серверу и в обратном направлении. Список атрибутов класса Variable представлен в табл. 20.

Адресное пространство сервера структурировано иерархически, однако узлы на разных ветвях иерархии могут иметь ссылки друг на друга, поэтому адресное пространство представимо в виде многосвязной сети узлов.

Все ссылки являются объектами класса ReferenceType.

Сервер может выделять из адресного пространства некоторую часть (View) для упрощения доступа клиента к некоторому набору данных.

Таблица 20

Список атрибутов узла класса Variable

Атрибут	Обязательный (M)/ необязательный (O)	Тип данных	Примечания
Атрибуты класса BaseNodeClass	—	См. табл. 13	—
Value	M	Определяется атрибутом DataType	Хранит последнее значение
DataType	M	NodeId	Хранит тип данных значения
ValueRank	M	Int32	Определяет, содержится ли в атрибуте Value число или массив: Если содержит: —3: Value может быть числом или одномерным массивом; —2: Value может быть как числом, так и массивом произвольной размерности; —1: Value — это число; 0: Value — массив с одним или несколькими измерениями; 1: Value — одномерный массив; n>1: Value — массив с n измерениями
ArrayDimensions	O	UInt32 []	Определяет максимально возможную длину массива для каждого измерения. Если максимум неизвестен, должно содержать 0. При ValueRank<=0 должно содержать null
AccessLevel	M	AccessLevelType	Показывает, доступен ли атрибут Value только для чтения или для чтения и записи
UserAccessLevel	M	AccessLevelType	Показывает атрибуты доступа (чтение/запись) с учетом прав пользователя
MinimumSamplingInterval	O	Duration	Содержит число (в мс), которое показывает, как часто сервер обновляет данное значение
Historyzing	M	Boolean	Показывает, сохраняет ли сервер значения

Окончание табл. 20

Атрибут	Обязательный (M)/ необязательный (O)	Тип данных	Примечания
AccessLevelEx	O	AccessLevelExType	Является расширенной версией атрибута AccessLevel (представляет собой 23-битовое число, первые 8 бит которого совпадают с значением атрибута AccessLevel)

6.3.4. Взаимодействие между клиентом и сервером

Взаимодействие между участниками обмена данными определено как набор служб (services). Данные службы могут быть сгруппированы в наборы (services sets).

Службы позволяют клиенту формировать запросы и получать ответы от сервера. Также службы позволяют клиенту подписаться на оповещения от сервера, с помощью которых сервер может сигнализировать о возникновении аварий, событий, изменении значений данных, а также о завершении работы программ.

Сообщения между клиентом и сервером могут быть закодированы как в двоичном, так и в гипертекстовом (XML) формате. В зависимости от этого они могут передаваться поверх существующего стека протоколов (например, поверх TCP- или HTTP-протокола).

Обмен данными происходит в рамках сессий, которые определяются как логическое соединение клиента и сервера. Механизм сессий не зависит от особенностей нижележащего протокола, при этом сессия не прерывается автоматически в случае разрывов нижележащих протоколов. Установление и завершение сессии выполняется с помощью запросов и ответов клиента и сервера.

Так как функциональность клиента и сервера может быть совмещена в одном устройстве, допускается существование комбинированного узла с функциями клиента и сервера.

Модель архитектуры клиента приведена на рис. 31.

Клиентское приложение использует набор функций (OPC UA API) для отправки и получения запросов и ответов от сервера в рамках концепции сервисов. Доступные сервисы описаны в стандарте.

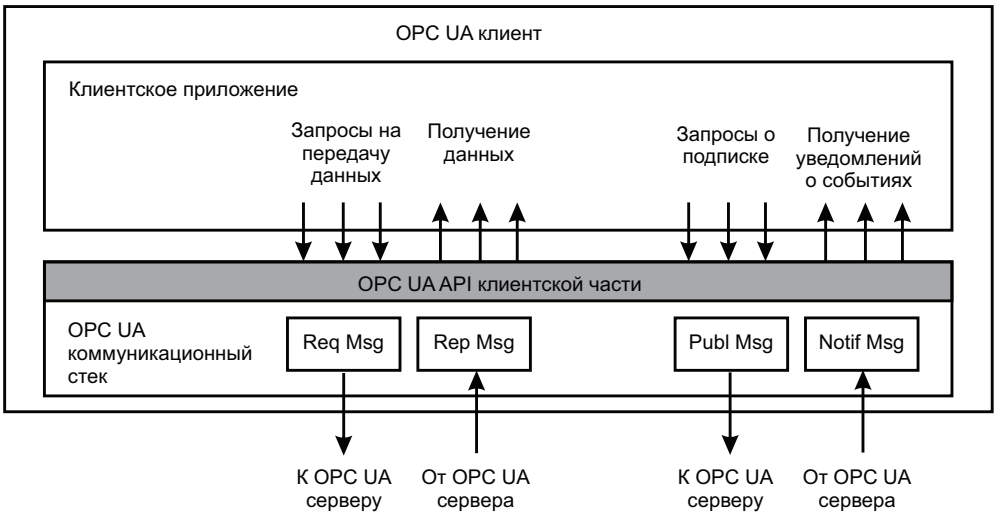


Рис. 31. Архитектура OPC UA-клиента

Модель архитектуры сервера представлена на рис. 32.

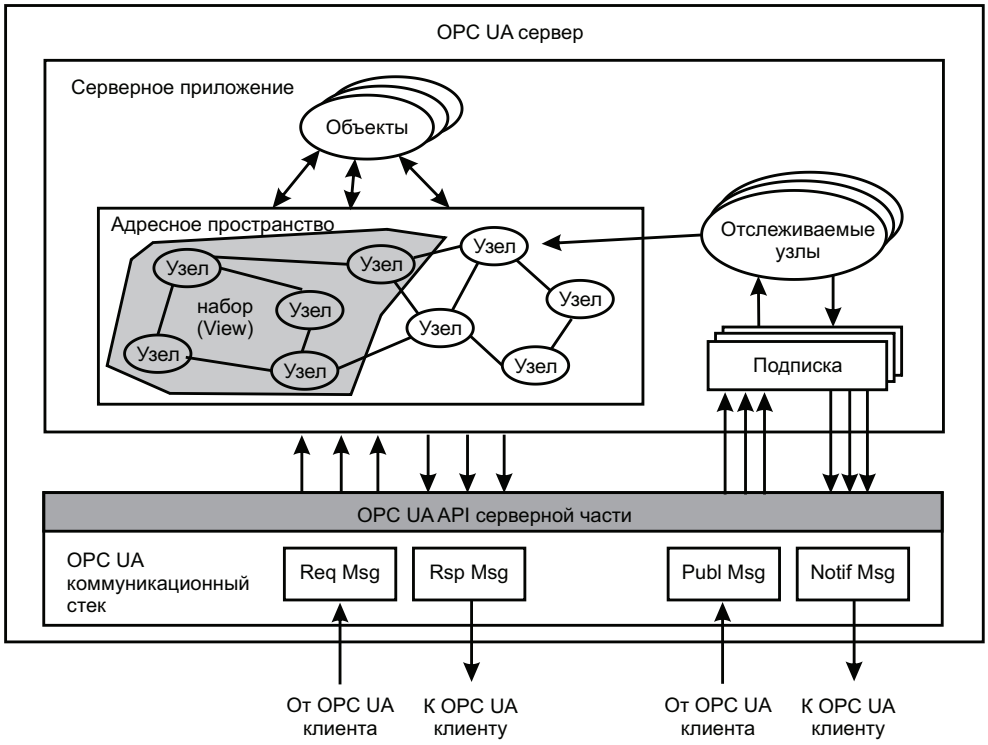


Рис. 32. Архитектура OPC UA-сервера

Из рис. 32 видно, что серверное приложение связано с объектом наблюдения, который может быть как аппаратным, так и программным объектом и располагаться как внутри, так и вне серверного приложения. Серверное приложение использует набор функций OPC UA API для отправки и получения сообщений (OPC UA Messages) от клиента. Адресное пространство сервера моделируется множеством узлов, доступных для клиента посредством сервисов (интерфейсов и методов). Связи между узлами позволяют серверу построить иерархическую, полносвязную или любую другую структуру.

Подмножество узлов может быть организовано в набор (view). Наборы позволяют ограничить количество видимых клиентом узлов. Отслеживаемые узлы (monitored item) создаются клиентом для отслеживания состояния узлов из адресного пространства. Отслеживаемые узлы могут обнаружить изменение данных или возникновение какого-либо события или тревоги. После чего они генерируют уведомление (notification), которое отправляется клиенту через подписку. Клиент может управлять подпиской для управления потоком событий от отслеживаемых объектов.

Взаимодействие между клиентом и сервером выполняется посредством сервисов. Все определенные в стандарте сервисы разделены на ряд наборов по их назначению:

- сервисы для определения доступных серверов (discovery service set). Эти сервисы позволяют найти доступные OPC-серверы;
- сервисы для формирования защищенного канала (secure channel service set);
- сервисы для формирования сеанса (session service set). Формируют сеанс на уровне приложений;
- сервисы для управления узлами (node management service set). Позволяют клиенту добавлять, изменять и удалять узлы в адресном пространстве сервера;
- сервисы управления наборами узлов (view service set);
- сервисы запросов (query service set). Позволяют клиенту получить доступ к узлам в адресном пространстве сервера;
- сервисы управления атрибутами (attribute service set). Позволяют считывать и записывать значения атрибутов, которые являются свойствами узла;
- сервисы для вызова методов (method service set). Позволяют вызывать функции, которые определены для объектов управления.

Это позволяет управлять объектами автоматизации через OPC-сервер;

- сервисы для работы с отслеживаемыми объектами (monitored item service set). Позволяют клиенту создавать отслеживаемые объекты и привязывать их к переменным, атрибутам или событиям;
- сервисы управления подписками (subscription service set). Позволяют клиентам создавать и управлять подписками, которые периодически отправляют клиенту уведомительные сообщения об отслеживаемых объектах. Подписки существуют относительно независимо от клиентов, и поэтому, как правило, в качестве основы для такого метода обмена берется транспортный протокол без установления соединения и без подтверждения доставки (UDP). Однако, если ни один из клиентов не обращается долгое время к подписке (не обновляет ее время жизни), подписка будет закрыта сервером и соединенные с ней отслеживаемые объекты удалены. Подписки имеют в своем составе поддержку обнаружения потерянных сообщений. Каждое уведомительное сообщение содержит сквозной порядковый номер, который позволяет клиенту обнаружить потерянные сообщения. Если клиент обнаружил потерю сообщения, он может запросить одно или несколько сообщений от сервера повторно.

6.3.5. Набор библиотек для работы с OPC UA

Так как стандарт OPC UA описан вплоть до байтового формата при передаче данных по сети, у разработчика взаимодействующего с OPC-приложения есть возможность написать весь механизм обмена с нуля. В качестве среды передачи для сообщений OPC может выступать установленное соединение по протоколу TCP или WebSocket. Единственным требованием, которое предъявляется стандартом OPC к протоколу-носителю сообщений, является возможность полнодуплексной передачи потока байт.

Однако, организацией OPC-Foundation уже разработан набор библиотек для нескольких популярных языков программирования (C++, C#, Java) [21], упрощающих работу с OPC-сообщениями.

Например, для подключения к проекту под платформу.NET данный набор доступен в виде сборок:

- Opc.Ua.Core.dll — набор базовых функций и типов;
- Opc.Ua.Client.dll — набор функций для создания клиентского приложения;
- Opc.Ua.Server.dll — набор функций для создания серверного приложения.

Рассмотрим пример реализации OPC-клиентом, написанным на языке C#, части, отвечающей за просмотр доступных серверов и точек подключения.

После подключения вышеописанных библиотек становятся доступными пространства имен Opc.Ua, Opc.Ua.Client, которые нужно указать директивой namespace:

```
using namespace Opc.Ua
using namespace Opc.Ua.Client
```

Первым этапом работы с OPC является получение списка OPC-серверов, запущенных на локальной машине. Для получения данного списка на локальной машине используется специальный сервер с уникальным именем LocalDiscoveryServer. Этот сервер доступен под одним или несколькими из следующих имен точек подключения:

opc.tcp:\\localhost:4840
https:\\localhost:443
opc.wss:\\localhost:443

Для получения списка серверов требуется создать в проекте C# объекты следующих типов:

```
ApplicationDescriptionCollection servers = null;
DiscoveryClient client;
```

Для указания адреса точки подключения необходимо также создать объект типа Uri:

```
Uri discoveryUrl = new Uri(sUrl);
```

где sUrl — объект типа string, содержащий адрес сервера.

Для получения списка серверов по протоколу TCP достаточно, чтобы аргумент sUrl выглядел как

opc.tcp:\\<имя узла>:<порт>

Например, для получения списка серверов на локальном узле необходимо присвоить

```
sUrl="opc.tcp:\\localhost:4840";
```

Вызов функции для получения списка:

```
client = DiscoveryClient.Create(Uri);
servers = client.FindServers(null);
```

В возвращаемой коллекции объектов можно обращаться к записи о конкретном сервере по индексу.

В каждой записи о сервере присутствует массив его DiscoveryEndpoints (точек подключения для обзора сервера). Обращение к этому массиву выполняется с помощью свойства DiscoveryUrls.

Например, получение первой Discovery-точки подключения первого найденного сервера будет иметь вид

```
sUrl = servers[0].DiscoveryUrls[0];
```

Зная такую точку подключения, можно получить все остальные точки подключения к данному серверу, используя вызов функции GetEndpoints() объекта client. Предварительно создаем объект endpoints типа EndpointDescriptionCollection для хранения информации о точках подключения:

```
EndpointDescriptionCollection endpoints = null;
Uri discoveryUrl = new Uri(sUrl);
client = DiscoveryClient.Create(Uri);
endpoints = client.GetEndpoints(null);
```

Таким образом, код на C#, получающий список точек подключения для локального сервера и помещающий его в объект ComboBox с именем UrlCB, приведен в следующем фрагменте (более полный код приведен в поставляемом OPC Foundation примере Sample OPC Client):

```
string sUrl="opc.tcp:\\localhost:4840";
Uri discoveryUrl = new Uri(sUrl);
ApplicationDescriptionCollection servers = null;
DiscoveryClient client;
EndpointDescriptionCollection endpoints = null;
client = DiscoveryClient.Create(Uri);
servers = client.FindServers (null);
for (int iServer = 0; iServer < servers.Count; iServer++)
{
```

```
    sUrl = servers[iServer].DiscoveryUrls[0];
    discoveryUrl = new Uri(sUrl);
    client = DiscoveryClient.Create(Uri);
    endpoints = client.GetEndpoints(null);
    for (int i = 0; i < endpoints.Count; i++)
    {
    UrlCB.Items.Add(endpoints[i].EndpointUrl);
    }
}
```

6.3.6. Установление соединения с OPC UA-сервером, используя приложение-клиент на C#

Для установления соединения необходимо создать объект сессии (тип Session из библиотеки Opc.Ua.Client). В свою очередь, для создания объекта сессии клиентом необходимо выполнение нескольких предварительных шагов:

Создание объекта, описывающего клиентское приложение (имеет тип ApplicationConfiguration):

```
//создание объекта configuration
ApplicationConfiguration configuration = new ApplicationConfiguration();
//задание имени приложения-клиента
configuration.ApplicationName = "UA Client";
//задание типа приложения;
//возможные типы представлены в таблице 21
configuration.ApplicationType = ApplicationType.Client;
//задание Uri приложения (актуально для серверных приложений)
configuration.ApplicationUri = "http://localhost/MyClient";
//задание дополнительного Uri
configuration.ProductUri = "http://Test/Education";
//создание объекта для хранения настроек безопасности
configuration.SecurityConfiguration = new SecurityConfiguration();
```

Возможные типы объектов, описывающих взаимодействие по протоколу OPC UA (в соответствии с библиотекой Opc.Ua)

Тип	Описание
Server	Приложение является OPC UA сервером для предоставления данных
Client	Приложение является OPC UA клиентом для сбора данных
ClientAndServer	Приложение выполняет функции OPC UA сервера и клиента одновременно
DiscoveryServer	Приложение является сервером, предоставляющим данные о доступных серверах на компьютере

Настройка сертификата клиента

```
//создание объекта класса CertificatIdentificator,
//содержащего информацию о сертификате
configuration.SecurityConfiguration.ApplicationCertificate =
new CertificateIdentifier();
//указание места хранения сертификата.
//Доступные места хранения представлены в табл. 22.
configuration.SecurityConfiguration.ApplicationCertificate
     .StoreType = CertificateStoreType.Windows;
//указание на размещение сертификата в общем хранилище
//сертификатов компьютера (LocalMachine) и в подхранилище с именем
     (My).
configuration.SecurityConfiguration.ApplicationCertificate
     .StorePath = "LocalMachine\\My";
//Строка для поиска сертификата в хранилище содержит имя приложения,
//которое его туда поместило. Это позволит находить этот сертификат
//в любой момент времени работы программы и не создавать дубликата
configuration.SecurityConfiguration.ApplicationCertificate
     .SubjectName = configuration.ApplicationName;
//добавляем настройку, которая настраивает доверие между всеми
//сертификатами на данном компьютере в персональном подхранилище
configuration.SecurityConfiguration.TrustedPeerCertificates
     .StoreType = CertificateStoreType.Windows;
configuration.SecurityConfiguration.TrustedPeerCertificates
     .StorePath = "LocalMachine\\My";
//поиск существующего сертификата для клиента (возможно мы уже
```

```
//запускали нашу программу, и сертификат уже создан)
clientCertificate  =  configuration.SecurityConfiguration
     .ApplicationCertificate.Find(true);
//если сертификат не найден, значит создаем его
if (clientCertificate == null)
{
//обычно сертификат создается не при запуске программы клиента,
//а однократно при ее установке на компьютер, но если
//по каким-то причинам сертификат не был создан, создаем его
//с привязкой к нашему приложению клиента
clientCertificate = CertificateFactory.CreateCertificate(
configuration.SecurityConfiguration.ApplicationCertificate
     .StoreType,
configuration.SecurityConfiguration.ApplicationCertificate
     .StorePath,
configuration.ApplicationUri,
configuration.ApplicationName,
null,
null,
1024,
120);
}
```

Указание места расположения сертификата (в соответствии с библиотекой Opc.Ua.Core):

Тип	Описание
Windows	Используется стандартное хранилище сертификатов, управляемое операционной системой. Этот вариант более защищен с точки зрения безопасности сертификатов
Directory	Сертификат хранится в выбранной разработчиком директории

Указание используемых протоколов транспортного уровня обмена

Для использования TCP-протокола необходимо указать следующую строку кода:

```
configuration.TransportConfigurations.Add(
new TransportConfiguration(Utils.UriSchemeOpcTcp,
typeof(Opc.Ua.Bindings.UaTcpBinding)));
```

Использование других транспортных протоколов задается аналогично. Например:

```
configuration.TransportConfigurations.Add(
new TransportConfiguration(Utils.UriSchemeHttp,
typeof(Opc.Ua.Bindings.UaSoapXmlBinding)));
```

Настройка параметров транспортного протокола

Данная настройка требуется для ограничения размеров и количества сообщений при обмене данными. Это позволяет защититься от DDOS-атак, связанных с переполнением очереди запросов к серверу или клиенту.

```
//создание объекта класса TransportQuotas, отвечающего
//за параметры транспортного протокола
configuration.TransportQuotas = new TransportQuotas();
//установить значение таймаута для разрыва сессии в миллисекундах
configuration.TransportQuotas.OperationTimeout = 360000;
//установить значение максимальной строки данных
configuration.TransportQuotas.MaxStringLength = 67108864;
//создать объект класса ServerConfiguration
configuration.ServerConfiguration = new ServerConfiguration();
```

Задать настройки клиента

```
//создать объект класса ClientConfiguration
configuration.ClientConfiguration = new ClientConfiguration();
//установить длительность таймаута сессии клиента
configuration.ClientConfiguration.DefaultSessionTimeout = 360000.
```

Проверка на соответствие выбранной конфигурации приложения

```
configuration.Validate(ApplicationType.Client);
```

Данный вызов позволяет проверить целостность всех параметров, а также настраивает некоторые внутренние переменные объекта configuration.

После создания объекта класса ApplicationConfiguration необходимо связать его с выбранной точкой подключения к выбранному серверу.

```
//создаем объект, описывающей точку подключения
EndpointDescription endpointDescription =
new EndpointDescription();
```

```
//присваиваем свойству EndpointUrl имя выбранной точки подключения
//(это имя можно получить с помощью функции GetEndpoints,
//описанной выше)
endpointDescription.EndpointUrl = Url;
//указываем режим безопасности нужной нам точки подключения
endpointDescription.SecurityPolicyUri = SecurityPolicies.None;
//указываем используемый алгоритм обмена ключами и шифрования
endpointDescription.SecurityMode = MessageSecurityMode.None;
//указываем используемый профиль транспортного уровня
endpointDescription.TransportProfileUri =
Profiles.WsHttpXmlOrBinaryTransport;
```

Ищем сертификат сервера в хранилище сертификатов

```
CertificateIdentifier certificateIdentifier = new CertificateIdentifier();
certificateIdentifier.StoreType = CertificateStoreType.Windows;
certificateIdentifier.StorePath = "LocalMachine\\My";
certificateIdentifier.SubjectName = "UA Test Client";
X509Certificate2 serverCertificate = certificateIdentifier.Find();
if (serverCertificate == null)
//если сертификат не найден – выдаем сообщение
//и завершаем программу с ошибкой
{
throw ServiceResultException.Create(
StatusCodes.BadCertificateInvalid,
"Could not find server certificate: {0}",
certificateIdentifier.SubjectName);
}
//указываем найденный сертификат сервера
endpointDescription.ServerCertificate =
          serverCertificate.GetRawCertData();
//создаем объект класса EndpointConfiguration (передавая
//в конструктор объект класса ApplicationConfiguration для того,
//чтобы из него были взяты значения по-умолчанию).
EndpointConfiguration endpointConfiguration =
          EndpointConfiguration.Create(configuration);
//устанавливаем значение таймаута для запросов в канале
endpointConfiguration.OperationTimeout = 300000;
//устанавливаем двоичный формат сообщений
```

```
endpointConfiguration.UseBinaryEncoding = true;
//создаем объект класса ConfiguredEndpoint, описывающий
//настроенную точку подключения
ConfiguredEndpoint endpoint = new ConfiguredEndpoint(
                                    null,
                                    endpointDescription,
                                    endpointConfiguration);
//создаем объект класса BindingFactory
BindingFactory bindingFactory = BindingFactory.Create(configuration);
//обновляем объект endpoint, используя точку подключения по умолчанию
if(endpoint.UpdateBeforeConnect)
{
endpoint.UpdateFromServer(bindingFactory);
endpointDescription = endpoint.Description;
endpointConfiguration = endpoint.Configuration;
}
```

Необходимо переопределить функцию, которая будет вызвана, если сертификат сервера не содержится среди доверенных сертификатов на компьютере, где запущен клиент. Она может всегда давать одобрение на использование сертификата (даже если он недоверенный — untrusted), как показано в примере, или может оставить этот вопрос на усмотрение пользователя непосредственно при попытке открытия сессии с сервером. Данная функция вызовется автоматически.

```
private void CertificateValidator_CertificateValidation(
                        CertificateValidator validator,
                        CertificateValidationEventArgs e)
{
    e.Accept = true;
}
//при настройке объекта configuration необходимо указать нашу
//функцию, переопределив свойство CertificateValidation
configuration.CertificateValidator.CertificateValidation +=
    new CertificateValidationEventHandler(
    CertificateValidator_CertificateValidation);
```

Заключительным этапом подготовки к установлению сессии между клиентом и сервером является создание объектов классов

SessionChanell и Session. При их создании и настройке используются сконфигурированные ранее объекты clientCertificate (системный класс X509Certificate2), configuration (класс ApplicationConfiguration, содержащий информацию о приложении-клиенте), endpoint (класс ConfiguredEndpoint, содержащий информацию о выбранной точке подключения).

```
SessionChannel channel = SessionChannel.Create(
configuration,
endpointDescription,
endpointConfiguration,
bindingFactory,
clientCertificate,
null);
Session m_Session = new Session (channel, configuration, endpoint);
m_Session.ReturnDiagnostics = DiagnosticsMasks.All;
//для обработки периодических событий-запросов о поддержании
//сессии в открытом состоянии необходимо добавить функцию
//Session_KeepAlive и указать ее в свойстве KeepAlive объекта
//m_Session
m_Session.KeepAlive +=
          new KeepAliveEventHandler(Session_KeepAlive);
```

Сама функция Session_KeepAlive может быть пустой:

```
private void Session_KeepAlive(Session session,
KeepAliveEventArgs e)
{
}
```

Общая схема порядка создания и настройки компонентов для OPC UA-клиента изображена на рис. 33.

Объект класса Session используется для открытия сессии с сервером, взаимодействия с сервером с целью получения и отправки данных, закрытия соединения.

Для установления соединения с сервером используется метод Open объекта класса Session:

```
UserIdentity identity = new UserIdentity();
m_Session.Open("Session Name", identity);
```

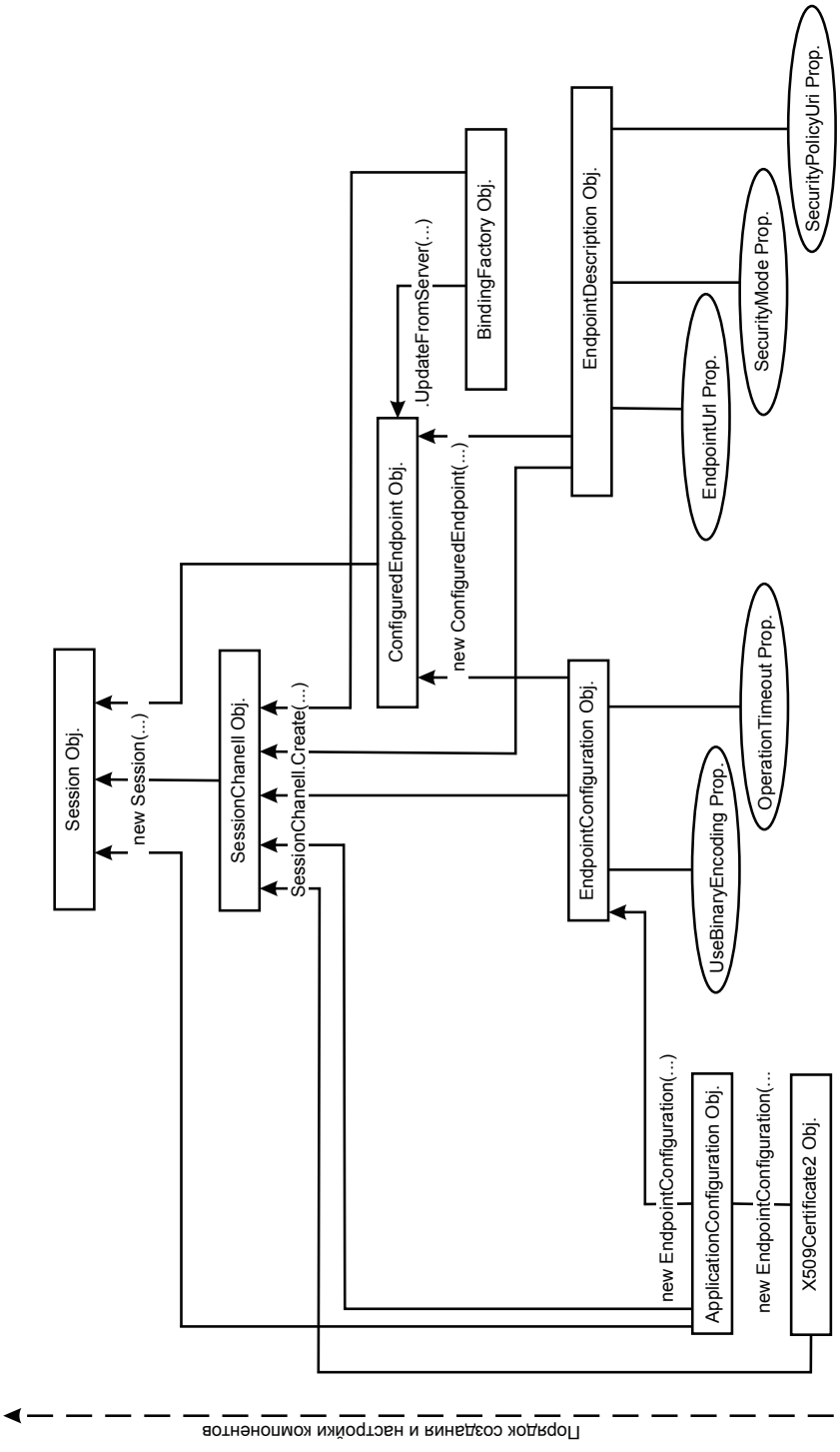


Рис. 33. Схема порядка создания и настройки компонентов для соединения с OPC UA-сервером.

6.3.7. Получение данных с OPC UA-сервера, используя приложение-клиент на C#

Для просмотра структуры адресного пространства сервера используется объект определенного в библиотеке Opc.Ua.Client.dll класса Browser (адресное пространство Opc.Ua.Client).

При создании объекта ему передается предварительно созданный объект класса Session, описывающий сессию между клиентом и сервером:

```
Browser m_Browser = new Browser(m_Session);
//настраивается фильтр, позволяющий получать только те узлы
//адресного пространства, которые организованы в иерархическую
//структуру
m_Browser.ReferenceTypeId = ReferenceTypes.HierarchicalReferences;
```

Для сохранения результатов просмотра необходимо объявить объект типа ReferenceDescriptionCollection и передать его в m_Brouser для создания и заполнения с помощью ключевого слова ref (передача по ссылке):

```
ReferenceDescriptionCollection browseResults = null;
```

С помощью метода Browse можно получить список дочерних узлов по отношению к заданному узлу nodeToBrowse.

```
browseResults = m_Browser.Browse(nodeToBrowse);
```

Заданный узел передается в метод Browse с помощью объекта nodeToBrowse класса NodeId. Для получения списка дочерних узлов корневого узла сервера необходимо создать объект класса NodeId с помощью специального аргумента конструктора класса:

```
nodeToBrowse = new NodeId(Objects.RootFolder, 0);
```

Если же требуется получить список дочерних узлов некорневого узла, то используется объект класса NodeId, возвращенный предыдущим вызовом метода Browse.

Пример обращения к серверу для получения списка дочерних узлов корневого узла иерархии:

```
try
{
    nodeToBrowse = new NodeId(Objects.RootFolder, 0);
    browseResults = m_Browser.Browse(nodeToBrowse);
}
catch(Exception e)
{
          throw e;
}
```

Результат, полученный в объекте browseResults содержит коллекцию элементов типа ReferenceDescription, каждый из которых описывает связь между узлами и содержит атрибуты данной связи. При описании связанного узла объект ReferenceDescription содержит поле NodeId, которым можно воспользоваться для передачи его в метод Browse и получения списка дочерних узлов текущего. Обход иерархической структуры адресного пространства можно сделать с помощью алгоритма обхода графа в ширину (BFS) с использованием класса Queue (очередь).

Пример кода, добавляющий полученный список всех узлов сервера в объект TVAddressSpace типа TreeView, имеет вид

```
Queue<TreeNode> Qu;
TreeNode TN, TNNew;
TVAddressSpace.Nodes.Clear();
m_Browser = new Browser (m_Session);
Qu = new Queue<TreeNode>();
m_Browser.ReferenceTypeId = ReferenceTypes.HierarchicalReferences;
TN = new TreeNode();
RC = new ReferenceDescription();
nodeToBrowse = new NodeId (Objects.RootFolder, 0);
RC. NodeId = nodeToBrowse;
TN. Text = "Root";
TN. Tag = RC;
TVAddressSpace.Nodes.Add(TN);
Qu.Enqueue(TN);
while(Qu.Count > 0)
{
```

```
TN = Qu.Dequeue();
nodeToBrowse = (NodeId)((ReferenceDescription)TN.Tag).NodeId;
browseResults = m_Browser.Browse(nodeToBrowse);
if (browseResults == null)
    for (i = 0; i < browseResults.Count; i++)
    {
          RC = browseResults[i];
          TNNew = new TreeNode();
          TNNew.Text = RC.BrowseName.Name;
          TNNew.Tag = RC;
          TN. Nodes.Add(TNNew);
          Qu.Enqueue(TNNew);
    }
}
```

Для считывания значений атрибутов узла используется метод Read объекта m_Session класса Session. Данному методу передается предварительно заполненная коллекция атрибутов, которые должны быть получены от сервера. Данная коллекция содержится в объекте класса ReadValieIdCollection. Результат работы метода будет помещен в коллекцию класса DataValueCollection, которая передается в метод по ссылке и создается им.

```
ReadValueIdCollection nodesToRead;
//здесь необходимо создание и заполнение структуры nodesToRead
……
//создание ссылки на объект, содержащий результаты запроса
DataValueCollection results=null;
//создание ссылки на объект, содержащий диагностическую информацию
DiagnosticInfoCollection diagnosticInfos = null;
m_Session.Read(null,0, TimestampsToReturn.Both, nodesToRead,
out results, out diagnosticInfos);
```

Заполнение объекта nodesToRead производится путем создания и добавления в коллекцию объектов типа ReadValueId, в каждом из которых указывается следующая информация:

- объект класса NodeId, описывающий узел, атрибут которого должен быть считан. Ссылка на данный объект может быть получена из свойства NodeId объекта класса ReferenceDescription, кото-

рый описывает связь в иерархии адресного пространства сервера (см., например, алгоритм получения адресного пространства сервера);

- уникальный ID атрибута. Значения этого параметра получаются с использованием значений перечисляемого типа Attributes. Например, если требуется ID атрибута DisplayName, то можно использовать вместо него константу Attributes.DisplayName. Аналогично для атрибута Value вместо ID используется константа Attributes.Value и т. д. Следует отметить, что набор атрибутов для узла зависит от его класса, который может быть получен из поля NodeClass структуры ReferenceDescription. Например, аттрибут Value может быть считан только для узла класса Variable. Набор всех доступных атрибутов приведен в табл. 20. Набор же атрибутов какого-либо конкретного класса можно посмотреть в спецификации протокола OPC.

Пример кода, считывающего атрибуты узла и помещающего их значения в объект ListView:

```
//пусть ссылка на структуру NodeId содержится в объекте RD класса
//ReferenceDescription.
int i;
string AttrName;
DataView DV;
//объект lVAttributes класса ListView будет содержать таблицу из
//двух колонок: название атрибута и его значение
nodesToRead = new ReadValueIdCollection();
//Добавляем атрибуты, общие для всех классов узлов
//Для упрощения работы создаем функцию addAttribute, которая
//помещает выбранный атрибут в структуру nodesToRead
addAttribute((NodeId)RD.NodeId, Attributes.NodeId, nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.NodeClass,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.BrowseName,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.DisplayName,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.Description,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.WriteMask, nodesToRead);
```

```
addAttribute((NodeId)RD.NodeId, Attributes.UserWriteMask,
          nodesToRead);
//добавляем запросы атрибутов, соответствующих только какому-либо
//конкретному класса узла. Класс узла определяем с помощью свойства
//NodeClass объекта RD
switch (RD.NodeClass)
{
//для примера даны только списки атрибутов классов Object и Variable
case NodeClass.Object:
addAttribute((NodeId)RD.NodeId, Attributes.EventNotifier,
nodesToRead);
break;
case NodeClass.Variable:
addAttribute((NodeId)RD.NodeId, Attributes.Value, nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.DataType,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.ValueRank,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.ArrayDimensions,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.AccessLevel,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.UserAccessLevel,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.MinimumSamplingInterval,
          nodesToRead);
addAttribute((NodeId)RD.NodeId, Attributes.Historizing,
          nodesToRead);
break;
//здесь нужно добавить списки атрибутов для остальных классов узлов.
}
results = null;

DiagnosticInfoCollection diagnosticInfos = null;
//обращение к серверу за значениями атрибутов
m_Session.Read(null,0, TimestampsToReturn.Both, nodesToRead,
out results, out diagnosticInfos);
//заполняем поля ListView полученными результатами
```

```
lVAttributes.Items.Clear();
for (i = 0; i < results.Count(); i++)
{
DV = results[i];
if (DV!= null)
{
//формируем текстовое название атрибута
switch (nodesToRead [i].AttributeId)
{
case Attributes.NodeId: AttrName = "NodeId"; break;
case Attributes.NodeClass: AttrName = "NodeClass"; break;
case Attributes.BrowseName: AttrName = "BrowseName"; break;
case Attributes.DisplayName: AttrName = "DisplayName"; break;
case Attributes.Description: AttrName = "Descrtiption"; break;
case Attributes.WriteMask: AttrName = "WriteMask"; break;
case Attributes.UserWriteMask: AttrName = "UserWriteMask"; break;
case Attributes.EventNotifier: AttrName = "EventNotifier"; break;
case Attributes.Value: AttrName = "Value"; break;
case Attributes.DataType: AttrName = "DataType"; break;
case Attributes.ValueRank: AttrName = "ValueRank"; break;
case Attributes.ArrayDimensions: AttrName = "ArrayDimensions"; break;
case Attributes.AccessLevel: AttrName = "AccessLevel"; break;
case Attributes.UserAccessLevel: AttrName = "UserAccessLevel";
break;
case Attributes.MinimumSamplingInterval:
AttrName = "MinimumSamplingInterval"; break;
case Attributes.Historizing: AttrName = "Historizing"; break;
default: AttrName = "Unknown Attribute"; break;
}
//создаем объект ListViewItem для добавления в ListView
ListViewItem LVI = new ListViewItem(AttrName);
if(DV.Value!=null)
LVI.SubItems.Add (DV.Value.ToString());
else
LVI.SubItems.Add("Not Value");
lVAttributes.Items.Add(LVI);
}
}
```

Вспомогательная функция для добавления запроса об атрибуте в объект класса ReadValueIdCollection:

```
void addAttribute(NodeId Node, uint ID,
ReadValueIdCollection nodesToRead)
{
ReadValueId RVI = new ReadValueId();
RVI.NodeId = Node;
RVI.AttributeId = ID;
nodesToRead.Add(RVI);
}
```

Использование DV. Value.ToString () для текстового представления значения атрибута информативно для многих типов атрибутов, но не для всех. Например, у атрибута NodeClass такая форма вывода выведет число (ID класса). Для более информативного вывода необходимо производить преобразование ID в текстовое название «вручную».

6.3.8. Передача данных в OPC UA-сервер, используя приложение-клиент на C#

Запись в атрибут Value клиентом своего значения возможна только для изменяемых (Writable) узлов. Узнать данный параметр узла можно по его атрибуту AccessLevel. Если данный атрибут имеет значение 1 (Readable), значит изменить атрибуты этого узла нельзя. Если он имеет значение 3 (Readable+Writable), то значение для данного узла можно изменить.

Для изменения значений необходимо вызвать метод Write объекта m_Session класса Session. Данному методу передается предварительно заполненная коллекция объектов класса WriteValue, которые содержат информацию об узле, атрибуте и новом значении. Данная коллекция содержится в объекте класса WriteValueCollection. Результат работы метода будет помещен в коллекцию класса StatusCodeCollection, которая будет содержать информацию о том, прошла ли запись успешно, или нет.

Для примера проиллюстрируем запись в значение атрибута Value для узла, ссылка на который содержится в объекте RD класса ReferenceDescription. Тип данных значения узла — Byte.

Новое значение берется из заранее введенного пользователем числа в текстовое поле объекта TextBox с именем tbNewValue.

```
StatusCodeCollection results;
DataValue DV;
//создаем объект типа DataValue для передачи серверу нового значения
//указываем тип данных значения
Variant variant = new Variant(Type.GetType ("Byte"));
DV = new DataValue (variant);
//указываем новое значение
DV. Value = Convert.ToByte(tBNewValue.Text);
results = null;
//создание ссылки на объект, содержащий диагностическую информацию
DiagnosticInfoCollection diagnosticInfos = null;
//создаем объект класса WriteValueCollection для передачи методу
//Write
WriteValueCollection valuesToWrite = new WriteValueCollection();
//создаем объект класса WriteValue для заполнения коллекции
WriteValue attributeToWrite = new WriteValue();
//передаем ссылку на узел, чей атрибут будем изменять
attributeToWrite.NodeId = (NodeId)RD.NodeId;
//будем перезаписывать значение атрибута Value
attributeToWrite.AttributeId = Attributes.Value;
//передаем ссылку на новое значение
attributeToWrite.Value = DV;
valuesToWrite.Add(attributeToWrite);
//обращение к серверу для изменения значения атрибута
m_Session.Write(null, valuesToWrite, out results,
out diagnosticInfos);
```

Следует отметить, что изменение значения атрибута Value узла OPC-сервера не всегда означает изменение значения ячейки ПЛК, связанного с этим узлом.

6.3.9. Получение данных по подписке от OPC UA-сервера, используя приложение-клиент на C#

Использование механизма подписки позволяет получать от сервера уведомления о изменении значений выбранных узлов, что освобождает клиента от циклического опроса ячеек сервера и отслеживания текущего значения каждой ячейки.

Создание объекта подписки начинается с конструирования объекта класса Subscrtiption, входящего в библиотеку Opc.Ua.Client (пространство имен Opc.Ua.Client):

```
Subscription m_Subscription = new
          Subscription(m_Session.DefaultSubscription);
m_Subscription.DisplayName = "My Subscription Name";
m_Subscription.PublishingEnabled = true;
//задание интервала для опроса
m_Subscription.PublishingInterval = 100
//задание интервала обновления времени жизни подписки
//10*UaRefreshRate = 5s если UaRefreshRate = 500 (мс)
m_Subscription.KeepAliveCount = 10;
//UaRefreshRate*100 = 50s если UaRefreshRate = 500;
m_Subscription.LifetimeCount = 100;
m_Subscription.MaxNotificationsPerPublish = 100;
//привязка подписки к объекту текущей сессии
m_Session.AddSubscription(m_Subscription);
//обращение к серверу для создания подписки
m_Subscription.Create();
```

Начиная с этого момента, клиент отправляет запросы на продление времени жизни подписки каждые 5 с. Если сервер не получит этих обновлений в течение 50 с, подписка будет уничтожена.

```
//задание метода, который будет вызван в случае получения
//уведомления от сервера
m_Session.Notification += new NotificationEventHandler(
Session_Notification);
```

Вызов метода Session_Notification будет выполняться системой в случае получения от сервера уведомляющего пакета с информацией об изменениях в отслеживаемых узлах (MonitoredItem). Объект

класса Subscription использует для управления подпиской на конкретное событие (значение конкретного узла) от сервера объекты класса MonitoredItem.

Для примера создадим отслеживаемое событие, возникающее при изменении значения узла, на который ссылается объект RD класса ReferenceDescription:

```
//создаем объект класса MonitoredItem
MonitoredItem m_Item = new MonitoredItem(
          m_Subscription.DefaultItem);
//указываем, что нужно отслеживать атрибут узла RD.NodeId
m_Item.StartNodeId = (NodeId)RD.NodeId;
//указываем, что нужно отслеживать только атрибут Value
m_Item.AttributeId = Attributes.Value;
m_Item.MonitoringMode = MonitoringMode.Reporting;
//задаем интервал, с которым OPC-сервер будет опрашивать источник
//данных данного узла (например, ячейку в памяти ПЛК), мс
m_Item.SamplingInterval = 100;
m_Item.QueueSize = 1;
m_Item.DiscardOldest = false;
//добавляем отслеживаемое событие в подписку
m_Subscription.AddItem(m_Item);
//обращение к серверу для сохранения изменений в подписке
m_Subscription.ApplyChanges();
//проверка результата добавления изменений в подписку
if (m_Item.Status.Error!=null &&
          StatusCode.IsBad(m_Item.Status.Error.StatusCode))
{
throw ServiceResultException.Create(
          m_Item.Status.Error.StatusCode.Code,
          "Creation of data monitored item failed");
}
```

Создадим пример метода обратного вызова Session_Notification для обработки уведомлений от сервера, в котором новое значение отображается в поле объекта TextBox. При этом, так как метод Session_Notification вызывается системой в другом потоке от того, в котором были созданы и функционируют все объекты пользовательского интерфейса, то изменение свойства Text объекта TextBox необходимо

выполнять с использованием функции-делегата и ее вызовом с помощью метода BeginInvoke:

```
//объявляем функцию-делегат для передачи методу BeginInvoke
public delegate void InvokeDelegate(string Str);
//описываем функцию для обратного вызова. Ее имя должно быть
//использовано при создании объекта подписки (см. выше)
private void Session_Notification(Session session,
          NotificationEventArgs e)
{
//создаем объект для хранения сообщения от сервера
NotificationMessage message = e.NotificationMessage;
if (message.NotificationData.Count == 0)
{
return;
}
//анализируем новые значения
foreach (MonitoredItemNotification datachange in
message.GetDataChanges(false))
{
//анализируем объект MonitoredItem
MonitoredItem m_Item =
e.Subscription.FindItemByClientHandle(datachange.ClientHandle);
if (monitoredItem == null)
{
          continue;
}
//вызываем функцию для изменения свойства Text объекта TextBox
tBMonitoredValue.BeginInvoke(new InvokeDelegate(InvokeMethod),
    new object[] {datachange.Value.ToString()});
}
}
//функция для изменения свойства Text объекта TextBox
public void InvokeMethod(string Str)
{
tBMonitoredValue.Text = Str;
}
```

Результатом выполнения данного кода будет активность со стороны OPC-сервера: раз в 100 мс сервер будет отправлять UDP-пакеты для клиента. В поле данных этого пакета будет последнее известное серверу значение атрибута Value данного узла. При поступлении пакета на вход сетевого интерфейса клиента ОС вызывает функцию Session_Notification, внутри которой происходит обработка полученной информации (помещение нового значения атрибута Value узла в поле Text компонента класса TextBox).

Контрольные вопросы

1. Какие стандарты из группы стандартов OPC существуют в настоящее время?
2. Какую структуру имеет OPC-сервер, соответствующий стандарту OPC DA?
3. Какие этапы включает в себя алгоритм взаимодействия OPC-клиента с OPC-сервером по стандарту OPC DA?
4. В чем заключаются преимущества стандарта OPC UA по сравнению с OPC DA?
5. Какие структурные элементы содержит OPC UA-сервер?
6. Какие протоколы используются на транспортном уровне стека OPS UA?
7. Что представляет собой получение данных по подписке с сервера OPC UA?

Библиографический список

1. Зимин В. В. Промышленные сети : учеб. пособие для студентов вузов / В. В. Зимин ; Министерство науки и высшего образования РФ. — Н. Новгород : Нижегородский государственный технический университет, 2006. — 252 с. — ISBN 5-93272-339-4.
2. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы : учебник для вузов / В. Г. Олифер, Н. А. Олифер ; Министерство науки и высшего образования РФ. — 3-е изд. — Санкт-Петербург : Питер, 2006. — 958 с.
3. Таненбаум Э. Компьютерные сети / Э. Таненбаум, Д. Уэзеролл ; Министерство науки и высшего образования РФ. — 5-е изд. — Санкт-Петербург : Питер, 2012. — 960 с.
4. Агуров П. Последовательные интерфейсы ПК. Практика программирования / П. Агуров ; Министерство науки и высшего образования РФ. — Санкт-Петербург : БХВ-Петербург, 2004. — 496 с. — ISBN 5-94157-468-1.
5. Старостин А. А. Технические средства автоматизации и управления : учеб. пособие / А. А. Старостин, А. В. Лаптева ; Министерство науки и высшего образования РФ. — Екатеринбург : Изд-во Урал. ун-та, 2015. 168 с.
6. Федоренко Д. Ю. Программирование клиентов OPC на C++ и C#. — URL: https://www.studmed.ru/fedorenko-dyu-programmirovanie-opc-klientov-na-c-i-c-chast-2-opc-hda_b87148adccf.html (дата обращения: 20.02.2020).
7. EIA standard RS-232-C: Interface between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange. Washington, USA: Electronic Industries Association, Engineering Department. 1969. — URL: http://bitsavers.trailing-edge.com/pdf/datapro/communications_standards/2740_EIA_RS-232-C.pdf (дата обращения: 20.02.2020).
8. Electronic Industries Association (1983). Electrical Characteristics of Generators and Receivers for Use in Balanced Multipoint Systems. EIA Standard RS-485. — URL: http://www.softelectro.ru/rs485_en.html (дата обращения: 20.02.2020).
9. ANSI/TIA-568.0-D, Generic Telecommunications Cabling for Customer Premises, Ed. D, 09-2015. — URL: https://blog.siemon.com/standards/ansitia-568-c-0-generic-telecommunications-cabling-for-customer-premises (дата обращения: 20.02.2020).
10. МСЭ-Т G.652 13.11.2016 г. СЕРИЯ G: СИСТЕМЫ И СРЕДА ПЕРЕДАЧИ, ЦИФРОВЫЕ СИСТЕМЫ И СЕТИ. Характеристики среды передачи и оптических систем. Волоконно-оптические кабели. Характеристики одномодового оптического волокна и кабеля. — URL: https://docplayer.ru/29229210-Mezhdunarodnyy-soyuz-elektrosvyazi-seriya-g-sistemy-i-sreda-

peredachi-cifrovye-sistemy-i-seti-harakteristiki-sredy-peredachi-volokonn-oopticheskie.html (дата обращения: 20.02.2020).
11. MODBUS organization, MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3. — URL: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf. (дата обращения: 20.02.2020).
12. PROFIBUS Specification. Normative Parts of PROFIBUS -FMS, -DP, -PA according to the European standard EN 50170 Volume 2. Edition 1.0, March 1998. — URL: http://atep.kpi.ua/files/pdf/profibus_protocol_1268914098.pdf (дата обращения: 20.02.2020).
13. PROFINET System Description. PROFIBUS Nutzerorganisation e.V. October 2014. Order Number 4.132. — URL: http://us.profinet.com/wp-content/uploads/2012/11/PROFINET_SystemDescription_ENG_2014_web.pdf (дата обращения: 20.02.2020).
14. Janssen, D.; Büttner, H. Real-time Ethernet: the EtherCAT solu-tion, Computing & Control Engineering Journal. 2004. Vol. 15. P. 16–21.
15. Jürgen Lange, Frank Iwanitz and Thomas Burke. OPC from Data Access to Unified Architecture. 4th rev. Edition / Lange Jürgen. Berlin: VDE VERLAG GMBH, 2010. ISBN 3-978-3-8007-3242-5.
16. OPC Data Access Custom Interface Specification. Version 2.05A. June 28, 2002. — URL: http://advosol.com/OpcSpecs/opcda205_cust.pdf (дата обращения: 20.02.2020).
17. OPC Data Access Automation Interface Standard. Version 2.02. February 4, 1999. — URL: https://www-bd.fnal.gov/controls/opc/OPC_DA_Auto_2.02_Specification.pdf (дата обращения: 20.02.2020).
18. OPC Unified Architecture Specification. Part 1: Overview and Concepts. Release 1.04. November 22, 2017. — URL: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts (дата обращения: 20.02.2020).
19. OPC Unified Architecture Specification. Part 2: Security Model Release 1.04, August 3, 2018. — URL: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-2-security-model/ (дата обращения: 20.02.2020).
20. OPC Unified Architecture Specification. Part 3: Address Space Model. Release 1.04. November 22, 2017. — URL: https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-3-address-space-model/ (дата обращения: 20.02.2020).
21. Build OPC UA .NET applications using .NET StandardLibrary. — URL: http://opcfoundation.github.io/UA-.NETStandard/ (дата обращения: 20.02.2020).

ТИТАЕВ АЛЕКСАНДР АНАТОЛЬЕВИЧ
Кандидат технических наук, доцент департамента информационных технологий и автоматики Института радиоэлектроники и информационных технологий УрФУ. Преподает курс «Информационные сети и технологии» в ИРИТ-РтФ.
Область научных интересов: исследование и разработка протоколов передачи данных, беспроводные сенсорные сети, подвижные робототехнические системы. Под руководством Титаева А. А. студенты проводят исследовательские работы в перечисленных выше областях.